

DISEÑO DE ALGORITMOS

Y SU CODIFICACIÓN EN LENGUAJE C

María Adriana Corona Nakamura
María de los Ángeles Ancona Valdez

$$\left(x + \frac{b}{2a}\right)^2$$

$$x + \frac{b}{2a} =$$

Lenguaje C

Mc Graw Hill

INCLUYE CD-ROM

Diseño de algoritmos y su codificación en lenguaje C

Diseño de algoritmos y su codificación en lenguaje C

María Adriana Corona Nakamura

Universidad de Guadalajara

María de los Ángeles Ancona Valdez

Universidad de Guadalajara

Revisión técnica

Miguel Orozco Malo

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Ciudad de México

Carlos Villegas Quezada

Universidad Iberoamericana, Ciudad de México



MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA
LISBOA • MADRID • NUEVA YORK • SAN JUAN • SANTIAGO • AUCKLAND
LONDRES • MILÁN • MONTREAL • NUEVA DELHI • SAN FRANCISCO
SÃO PAULO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

Director Higher Education: Miguel Ángel Toledo Castellanos
Editor sponsor: Pablo E. Roig Vázquez
Coordinadora editorial: Marcela I. Rocha Martínez
Editora de desarrollo: Ana L. Delgado Rodríguez
Supervisor de producción: Zeferino García García

DISEÑO DE ALGORITMOS Y SU CODIFICACIÓN EN LENGUAJE C



A *Subsidiary of* ***The McGraw-Hill*** *Companies, Inc.*
Edificio Punta Santa Fe
Prolongación Paseo de la Reforma 1015, Torre A
Piso 17, Colonia Desarrollo Santa Fe,
Delegación Álvaro Obregón
C.P. 01376, México, D.F.
Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

ISBN: 978-607-15-0571-2

1234567890 1098765432101

Impreso en México *Printed in Mexico*

Contenido



Prefacio

Es sabido que la interacción que tenemos con la computadora se da por medio de programas. La finalidad de este texto es que el lector sea capaz de elaborar programas a la medida de cualquier problema que enfrente, sin importar el área de aplicación. Lo anterior puede parecer complicado, sobre todo para los alumnos de las carreras que no son afines, incluso hay quienes llegan a considerar la materia como poco importante para su formación. Por ello, en este libro queremos demostrar que:

- Programar es una tarea fácil.
- La práctica fortalece la habilidad de programar.
- La programación es útil para todas las áreas del conocimiento.

El razonamiento de las computadoras es diferente al de los seres humanos, es por ello que a quienes comienzan a programar les resulta una tarea difícil. El primer paso es no desesperarse, después se debe entender cómo razonan los humanos y posteriormente analizar cómo lo haría una computadora. Es importante hacer hincapié en que la parte más compleja de este proceso es el desarrollo de un algoritmo (diagrama de flujo o pseudocódigo), ya que constituye la parte lógica. Codificar, independientemente del lenguaje, es simplemente trascribir un algoritmo al lenguaje respectivo. El concepto de algoritmo lo utilizamos, en general, todas las disciplinas basadas en las matemáticas y la física, por ende en la programación, y es la etapa previa a la codificación.

Para llevar a cabo un programa necesitamos de un lenguaje de programación. Existen diferentes tipos, y en este libro nos enfocaremos específicamente en el lenguaje C estándar. También hay una gran variedad de compiladores, aquí utilizaremos tres de ellos: Turbo C, DEV-C++ y CodeBlocks; que son los más utilizados actualmente. Cabe destacar que todos los ejercicios presentados en el texto fueron probados en los compiladores antes citados.

En este trabajo se pensó en manejar de manera paralela el diagrama de flujo, el pseudocódigo y el lenguaje C, debido a la dificultad que existe entre los alumnos para realizar la conversión de la parte algorítmica al código (lenguaje C en nuestro caso).

Diseño de algoritmos y su codificación en lenguaje C introduce los conceptos fundamentales de la programación, especialmente en el diseño de algoritmos, la programación estructurada y la codificación en lenguaje C. Con esta obra el lector recibe una orientación básica y fundamental en dicha área.

El texto sigue las reglas clásicas de construcción de diseño de programas estructurados. Enfatiza particularmente el diseño de algoritmos, mostrándolos en sus múltiples expresiones: lenguaje natural, pseu-

docódigo, diagramas de flujo y lenguaje de programación. Asimismo, aborda la estructura de datos e incluye gran cantidad de ejemplos explicados y ejercicios, con el objetivo de que el lector pueda comprobar su progreso de aprendizaje.

El libro está estructurado en siete capítulos y un apartado de anexos. Los capítulos tienen una organización similar. Inician con una definición de conceptos teóricos y culminan con un conjunto de ejemplos y ejercicios prácticos que el lector debe realizar para afianzar sus conocimientos. El alumno puede transitar fácilmente en el texto gracias a la incorporación de elementos de apoyo para entender y aprender los diferentes términos que se manejan.

En el sentido conceptual, el primer capítulo presenta una amplia explicación de los algoritmos como técnica para describir los pasos ordenados y finitos que permiten resolver un problema o tarea específica, así como la diferencia entre algoritmos computacionales y algoritmos no computacionales. De igual manera, se definen y explican de forma sencilla otros conceptos básicos de la programación, entre los que destacan los lenguajes de programación y las etapas o pasos necesarios para la creación de programas de computadora.

El principal insumo para un programa de computadora son los datos, puesto que a partir del procesamiento de los mismos se obtienen diferentes tipos de información, que es básicamente el objetivo final de dichos programas. En este sentido, el segundo capítulo se centra en el análisis de los diferentes tipos de datos que existen y la forma de trabajar con ellos, de acuerdo con su naturaleza en el desarrollo de los programas. Con ejemplos muy claros y concisos se explican los tipos de datos, cómo se deben declarar las variables, el uso de las constantes, así como el papel tan importante que tienen los operadores y el buen manejo de estos al relacionar los datos dentro de las expresiones.

El tercer capítulo está dedicado a la explicación exhaustiva de la programación estructurada como técnica de desarrollo de programas, con énfasis en sus bondades; es fácil escribir, verificar, leer y dar mantenimiento a los programas. De igual forma, este capítulo señala los puntos principales relacionados con la programación estructurada, tales como la estructura de datos, el diseño descendente, la programación modular y, particularmente, en tres estructuras de control (secuencial, selectiva y repetitiva).

En el cuarto capítulo se plantea la importancia de hacer un uso eficiente de los recursos mediante el manejo de arreglos que permitan el ordenamiento y estructura de datos con la definición exacta de la memoria requerida. Dichos arreglos se clasifican, de acuerdo con las necesidades y requerimientos del programa, en dos tipos: unidimensionales y multidimensionales.

El capítulo cinco refiere a la programación modular como un método para mejorar la productividad de los programas. Siendo ésta una solución para resolver problemas complejos, la cual tiene como propósito dividir el problema en partes más pequeñas, de tal manera que se conforman módulos que son más sencillos de resolver por separado. Este método, entre otras ventajas, ofrece la facilidad de mantenimiento de los programas, la detección de errores y la reutilización de código. Inicialmente analizamos las funciones definidas por el usuario y al final las funciones predefinidas por el lenguaje C.

En el capítulo seis manejamos el tema de registros o estructuras; los cuales, a diferencia de un arreglo, nos permiten almacenar datos de diferente tipo en una sola variable. Comenzamos con la definición de un registro, después remarcamos las ventajas de definir nuestros propios tipos de datos, para accesar a las estructuras; por último los arreglos de estructuras y las estructuras anidadas.

En el capítulo siete vemos cómo los métodos de ordenamiento nos permiten clasificar datos y con los métodos de búsqueda encontramos un elemento en un arreglo; para terminar con los apuntadores. La memoria de una computadora puede verse como un vector de valores, donde cada una de sus posiciones es referenciada por un número hexadecimal llamado DIRECCIÓN. Los apuntadores o punteros son variables cuyo contenido es una dirección de memoria.

En el anexo A nos damos cuenta que Lenguaje C no tiene palabras reservadas para la entrada/salida estándar del sistema (teclado y pantalla). Estas operaciones se realizan mediante funciones de biblioteca que encontramos en el archivo de cabecera *stdio.h.* La E/S con *formato* se realiza por medio de las funciones *scanf() y printf.* Las *f* de scanf() y de printf() significan "con formato". En el caso de E/S de caracteres y cadenas vemos la conveniencia de utilizar la E/S sin formato.

Actualmente existen varios compiladores del lenguaje, entre los más utilizados encontramos Turbo C, DEV-C++ y Code::Blocks. El turbo C surgió en el sistema DOS como un compilado idóneo para los alumnos que iniciaban a programar por la gran ayuda que contenía, posteriormente surgió la versión 4.5 para Windows, pero lamentablemente tanto la versión libre como la que existía bajo licencia salieron del

mercado. Existen el DEV-C++ y el Code::Blocks, ambos son software libre y se utilizan en la plataforma Windows; la desventaja del primero es que la última versión fue la 4.9.9.2 (febrero de 2005); la ventaja de CodeBlocks es que también es compatible con Linux y la última versión fue la 10.05 (mayo de 2010). Las diferencias más importantes las encontramos en el anexo B.

Las aplicaciones ponen de manifiesto lo aprendido en el presente libro; la programación se puede aplicar en las diferentes áreas de la ingeniería y ciencias exactas; es por ello que elegimos aplicaciones en las áreas de la química, la ingeniería en comunicaciones y electrónica; la ingeniería mecánica, la ingeniería civil y topografía. En el anexo C apreciamos la codificación de las mencionadas aplicaciones.

En el anexo D podemos consultar el código ASCII, el cual resulta ser muy útil, sobre todo cuando tenemos problemas con la configuración del teclado.

Un glosario de términos y de funciones de lenguaje C es bastante útil cuando comenzamos a programar; por ello incluimos uno en el anexo E.

Asimismo, la obra se apoya en un CD, el cual contiene los siguientes recursos:

1. Aplicaciones.

   Los métodos numéricos son bastante utilizados en las diferentes ingenierías y áreas afines, por lo que desarrollamos los programas de métodos que consideramos más importantes.

   Al final del curso a los alumnos se les pide un proyecto donde apliquen lo aprendido durante el ciclo respectivo, seleccionamos por lo menos uno de cada área.

2. Los compiladores DEV-C++ y Code::Blocks; así como la instalación de la librería conio2 para DEV-C++ y Codeblocks.

   Se incluyen los Compiladores DEV-C++ y Code::Blocks, ya que son software libre.

   Adicionalmente describimos la instalación de la librería conio2, ya que el conio normal de ambos no incluye las funciones de Borland C, como son: clrscr(), gotoxy(), textcolo(), etc.

3. Ejercicios complementarios.

   En los ejercicios por completar el alumno será capaz de entender el algoritmo respectivo con base en el enunciado. En las pruebas a mano o de escritorio el alumno comprenderá cómo trabaja internamente la computadora. En el caso de los ejercicios abiertos de lenguaje C las versiones presentadas no son únicas, sólo le darán al alumno una idea de cómo puede crear su propia versión.

4. Salida de los ejercicios resueltos.

   Dichas salidas se presentan para que el alumno aprecie las posibles salidas que puede obtener después de compilar los respectivos programas.

La solución de todos los ejercicios incluidos en el CD está protegida ya que el propósito es que el alumno los capture, analice el algoritmo, vea la salida y posteriormente diseñe su propia versión.

Como hemos visto el CD es de gran utilidad para reforzar los conocimientos adquiridos en cada uno de los temas presentados en el libro.

Sobra decir que el texto es un valioso material de apoyo, sobre todo en los modelos educativos que favorecen el autoaprendizaje, puesto que lleva al lector paso a paso de los conceptos más básicos a procesos complejos, con el apoyo de ejemplos que refuerzan su comprensión y con ejercicios que propician la práctica y aplicación progresiva del conocimiento adquirido. Su publicación es bienvenida porque contribuye a llenar uno de los vacíos que existen en la enseñanza de esta disciplina.

Agradecemos de manera especial a Suhey Ayala Ramírez por su colaboración en esta edición.

María Adriana Corona Nakamura
María de los Ángeles Ancona Valdez



Capítulo 1

Conceptos básicos

Al término de este capítulo, el alumno será capaz de

- Identificar, describir y diferenciar conceptos básicos de la programación; así como comprender la importancia de las etapas para la elaboración de programas.

Contenido



1.1 Definición de lenguaje de programación

El lenguaje de programación es la combinación de símbolos y reglas que permiten la elaboración de programas con los cuales la computadora puede realizar tareas o resolver problemas de manera eficiente.

Los lenguajes de programación se clasifican en:

1. *Lenguaje máquina*. Las instrucciones son directamente entendibles por la computadora y no necesitan traductor para que la CPU (unidad de procesamiento central) pueda entender y ejecutar el programa. Utiliza un código binario (0 y 1), se basa en bits (abreviatura inglesa de dígitos binarios).
2. *Lenguaje de bajo nivel (ensamblador)*. Las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos.
3. *Lenguaje de alto nivel*. Es semejante al lenguaje humano (en general en inglés), lo que facilita la elaboración y comprensión del programa. Por ejemplo Basic, Pascal, Cobol, Fortran, C, etcétera.



1.2 Definición de algoritmo

Se denomina *algoritmo* al conjunto de pasos ordenados y finitos que permiten resolver un problema o tarea específica. Los algoritmos son independientes del lenguaje de programación y de la computadora que se vaya a emplear para ejecutarlo.

Todo algoritmo debe ser:

1. *Finito* en tamaño o número de instrucciones (tiene un primer paso y un último paso) y tiempo de ejecución (debe terminar en algún momento). Por lo tanto, debe tener un punto particular de inicio y fin.
2. *Preciso*. Debe tener un orden entre los pasos.
3. *Definido*. No debe ser ambiguo (dobles interpretaciones); si se ejecuta el mismo algoritmo el resultado siempre será el mismo, sin importar las entradas proporcionadas.
4. *General*. Debe tolerar cambios que se puedan presentar en la definición del problema.

Toda actividad que realizamos la podemos expresar en forma de algoritmo. Existen dos tipos de algoritmos, los que se desarrollan para ser ejecutados por una computadora, llamados *algoritmos computacionales*, y los que realiza el ser humano, es decir, *algoritmos no computacionales*; como ejemplos de éstos tenemos:

1. Cambiar un neumático (llanta) de un automóvil.
2. Preparar unos "huevos a la mexicana".
3. Calcular el área de un triángulo.

Ejemplo



Un algoritmo para cambiar el neumático desinflado de un automóvil

1. Inicio[1].
2. Bajar la herramienta y el neumático (llanta) de repuesto del automóvil.
3. Aflojar los birlos del neumático pinchado.
4. Acomodar el gato.
5. Levantar el automóvil.
6. Quitar los birlos del neumático desinflado.
7. Quitar el neumático desinflado.
8. Colocar el neumático de repuesto.
9. Fijar los birlos del neumático de repuesto.
10. Bajar el automóvil.
11. Apretar en forma definitiva los birlos del neumático de repuesto.

(*continúa*)

[1] Existen autores que en este tipo de algoritmos (no computacionales) *no* utilizan el *Inicio* y el *Fin*, ambos son opcionales.

(*continuación*)

12. Quitar el gato.
13. Guardar el neumático desinflado y la herramienta.
14. Fin.

Si revisamos, este algoritmo es finito (tiene 12 pasos[2]) y tiene un orden.

Ejemplo

Un algoritmo para preparar unos "huevos a la mexicana"

1. Poner la sartén en la estufa.
2. Poner aceite en la sartén.
3. Encender la estufa.
4. Cortar cebolla, tomate y chile en pedazos pequeños.
5. Poner la cebolla, el tomate y el chile en la sartén.
6. Abrir los huevos y verterlos en un recipiente.
7. Batir los huevos.
8. Poner los huevos batidos en la sartén.
9. Revolver la cebolla, tomate y el chile con los huevos hasta que queden estos últimos cocidos.
10. Vaciarlos en un plato.

Este algoritmo también es finito (tiene 10 pasos) y algunos pasos pueden estar en otro orden, por ejemplo los cuatro primeros puntos pudieron estar en un orden diferente y seguiríamos teniendo el mismo resultado.

Ejemplo



Un algoritmo para calcular el área de un triángulo

1. Inicio.
2. Solicitar (leer) los datos (la base y la altura).
3. Multiplicar la base por la altura y el resultado dividirlo entre dos, y guardarlo en una variable.
4. Mostrar (imprimir) el resultado almacenado en la variable.
5. Fin.

Al igual que en los dos ejemplos anteriores, se cumplen todas las características, solamente que este último algoritmo no es una situación cotidiana de la vida sino un cálculo específico el cual tiene un resultado exacto, o sea un valor.

1.3 Definición de programa de computadora

Existen diferentes conceptos; sólo mencionaremos tres:

1. Es un algoritmo desarrollado en un determinado lenguaje de programación, para ser utilizado por la computadora; es decir, es una serie de pasos o instrucciones ordenadas y finitas que pueden ser procesadas por una computadora, a fin de permitirnos resolver un problema o tarea específica.
2. Secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los datos que se desee procesar en la computadora.
3. Expresión de un algoritmo en un lenguaje preciso que puede llegar a entender una computadora.

[2] Sin considerar inicio y fin.

Como se comentó en la sección anterior, *no* todo algoritmo puede llegar a ser programa de computadora, debido a que existen algunos algoritmos que requieren ser realizados físicamente. Los programas que puede ejecutar una computadora son más de carácter de proceso lógico, por ejemplo el tercer algoritmo en el cual se realizan cálculos respectivos.

1.4 Etapas o pasos en la creación de un programa

Las fases para la creación de un programa son siete, aunque para algunos autores pueden describirse en sólo seis, pues omiten la primera porque es una etapa algo obvia. Las etapas se describen a continuación.

1.4.1 Definición del problema

Esta fase la proporciona el enunciado del problema, el cual requiere una definición clara y precisa (no debe ser ambiguo). Es importante que se entienda perfectamente lo que pretendemos que haga la computadora para poder continuar con la siguiente etapa.

1.4.2 Análisis del problema

Una vez que se ha comprendido lo que se desea que la computadora haga, la etapa de análisis es muy importante ya que en ésta se identifican tres factores indispensables:

1. Qué información se necesita para obtener el resultado deseado (datos de entrada).
2. Qué información se desea producir (datos de salida).
3. Los métodos y fórmulas que se necesitan para procesar los datos y producir esa salida.

1.4.3 Diseño y técnicas para la formulación de un algoritmo

La etapa de diseño se centra en desarrollar el algoritmo basándonos en las especificaciones de la etapa del análisis; podemos representar un algoritmo mediante el *diagrama de flujo* o el *pseudocódigo*.

Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un algoritmo; dicha representación gráfica se lleva acabo cuando varios símbolos (que indican diferentes procesos en la computadora) se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar las instrucciones para obtener los resultados deseados. Los símbolos utilizados han sido reglamentados por el Instituto Nacional de Normalización Estadounidense (ANSI, *American National Standards Institute*) y los apreciamos en la tabla 1.1.

Características de los diagramas de flujo:

- Todo diagrama debe tener un inicio y un fin.
- No se especifica la declaración de variables.
- Se deben usar solamente líneas de flujo horizontales y/o verticales.
- Se debe evitar el cruce de líneas utilizando los conectores.
- Se deben usar conectores sólo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
- Se debe evitar la terminología de un lenguaje de programación o máquina.
- Los comentarios se deben utilizar ya sea al margen o mediante el símbolo gráfico *comentarios* para que éstos sean entendibles por cualquier persona que lo consulte.
- Si el diagrama abarca más de una hoja es conveniente enumerarlo e identificar de dónde viene y a dónde se dirige.
- Sólo los símbolos de decisión pueden y deben tener más de una línea de flujo de salida.

Tabla 1.1 Símbolos gráficos más utilizados para dibujar diagramas de flujo de algoritmos

Símbolo	Descripción
	Inicio y final del diagrama de flujo.
	Entrada (leer) y salida de datos (imprimir).
	Símbolo de decisión. Indica la realización de una comparación de valores.
	Símbolo de proceso. Indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.
	Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.
Desde (i←VI, i<=VF, Inc)	Repetitiva desde número de iteraciones o repeticiones.
	Impresión[3]
	Conectores

Pseudocódigo

El pseudocódigo es la combinación del lenguaje natural (español, inglés o cualquier otro idioma), símbolos y términos utilizados dentro de la programación. Se puede definir como un lenguaje de especificaciones de algoritmos.

El pseudocódigo se creó para superar las dos principales desventajas del diagrama de flujo: es lento de crear y difícil de modificar sin un nuevo redibujo. Por otra parte, el pseudocódigo es más fácil de utilizar ya que es similar al lenguaje natural. Al contrario de los lenguajes de programación de alto nivel no tiene normas que definan con precisión lo que es y lo que no es pseudocódigo, por lo tanto varía de un programador a otro.

En este documento todo pseudocódigo lo iniciaremos con la palabra reservada *principal* para especificar la función denominada (*main*) en lenguaje C. Todo programa, al igual que toda función, debe contener las palabras reservadas de inicio ({) y fin (}) delimitando las instrucciones.

1.4.4 Codificación

En la etapa de codificación se transcribe el algoritmo definido en la etapa de diseño en un código reconocido por la computadora; es decir, en un lenguaje de programación; a éste se le conoce como código fuente. Por ejemplo el lenguaje "C" es un lenguaje de programación y es el que utilizaremos en el presente curso.

---

[3] Algunos autores utilizan el símbolo para salida de impresora; nosotros lo utilizaremos para salida de datos, a fin de diferenciarla de la entrada, ya que es válido en ambos casos.

1.4.5 Prueba y depuración

La prueba consiste en capturar datos hasta que el programa funcione correctamente. A la actividad de localizar errores se le llama *depuración*. Existen dos tipos de pruebas: de sintaxis y de lógica.

Las *pruebas de sintaxis* se ejecutan primero, son las más sencillas y las realiza el compilador del programa cada vez que se ejecuta el programa hasta que el código no presente errores, es decir que la sintaxis que requiere el lenguaje sea la correcta, de lo contrario el propio compilador va mostrando los errores encontrados para que se modifiquen y se pueda ejecutar el código; estos errores pueden ser falta de paréntesis, o puntos y comas o palabras reservadas mal escritas.

Las *pruebas de lógica* son las más complicadas ya que éstas las realiza el programador; consisten en la captura de diferentes valores y revisar que el resultado sea el deseado, es decir el programador tendría que modificar el código hasta que el programa funcione correctamente.

1.4.6 Documentación

Es la guía o comunicación escrita que permite al programador o al usuario conocer la funcionalidad del programa.

La documentación sirve para que el código fuente sea más comprensible para el programador o para otros programadores que tengan que utilizarlo, así como para facilitar futuras modificaciones (mantenimiento).

Hay dos tipos de documentación:

- *Interna.* Se generan en el mismo código y generalmente es mediante comentarios.
- *Externa.* Son los manuales y es independiente al programa. También puede ser la ayuda en el mismo software.

1.4.7 Mantenimiento

Se dice que un programa no se termina al 100%, ya que es necesario hacer algún cambio, ajuste o complementación para que siga funcionando correctamente; para llevarlo a cabo se requiere que el programa esté bien documentado.

Todos los programas tienen actualizaciones, por lo que surgen versiones diferentes. Por ejemplo: Windows 3.11, 95, 98, 2000, Millennium, Xp, Vista y 7.





Los datos y las operaciones básicas

Al término de este capítulo, el alumno será capaz de

- Identificar y utilizar los diferentes elementos que integran un programa: identificadores, constantes, variables, tipos de datos, operadores, expresiones, comentarios, palabras reservadas.

Contenido

2.1 Identificador

Un identificador es una secuencia de caracteres alfabéticos, numéricos y el guión bajo. Con ellos podemos dar nombre a variables, constantes, tipos de dato, nombres de funciones o procedimientos, etcétera.

Cada lenguaje de programación tiene sus propias características del tamaño del identificador; el estándar de lenguaje C no especifica un límite de tamaño para un identificador, pero para ciertas implementaciones de C++ sólo los primeros 31 caracteres son significativos (ANSI C). El programador tiene libertad para darle cualquier nombre a un identificador, siguiendo estas reglas:

1. Debe comenzar con una letra (A a Z) mayúscula o minúscula y no puede contener espacios en blanco. En *lenguaje C*, el carácter "_" (guión bajo) es considerado como letra, por lo que se puede utilizar como primer carácter.
2. El lenguaje C distingue mayúsculas de minúsculas porque tienen diferente código ASCII.
3. Letras, dígitos y el carácter guión bajo están permitidos después del primer carácter.
4. No pueden existir dos identificadores iguales, es decir, dos elementos de un programa no pueden nombrarse de la misma forma. Sin embargo, un identificador puede aparecer más de una vez en un programa.
5. No se puede utilizar una palabra reservada como identificador, sin embargo, los identificadores estándar se pueden redefinir.
6. En l*enguaje C* existen identificadores que podrían tener uno o varios puntos, tales como: persona.apellidoPaterno

   El punto indica el acceso a un campo de una estructura.

Sugerencias:

1. El identificador debe tener un nombre que sea *significativo*, es decir, que dé una idea de la información que almacena.
2. No utilizar nombres muy largos, es mejor utilizar abreviaturas estándar para que éstos tengan una longitud razonable.
3. En lenguaje C es usual escribir variables en minúscula, dejando las mayúsculas para las constantes. En los casos de nombres compuestos se suele poner la inicial de la segunda palabra en mayúscula.

Ejemplo



totalAlumnos, areaCirculo, numeroPositivo.

Identificadores válidos: numero, year2008, Base_1, funcion, division

Identificadores no válidos: número, ?precio, año, 2007, 4semestre.

En los tres primeros ejemplos se utilizan caracteres especiales y en los dos últimos el primer carácter no es letra. Hay que recordar que la ñ en el código ASCII, NO aparece con las demás letras.



2.2 Tipos de datos

Los diferentes objetos de información con los que un *algoritmo o programa* trabaja se conocen colectivamente como datos. Todos los datos tienen un tipo asociado con ellos; el tipo de un dato es el conjunto (rango) de valores que puede tomar durante el programa. Por ejemplo el tipo de un dato determina la naturaleza del conjunto de valores que puede tomar una variable. En la tabla 2.1 apreciamos que cada tipo de dato en lenguaje C está delimitado por un rango de valores. Como veremos en los ejemplos resueltos, si utilizamos valores fuera del rango correspondiente, el compilador no sabrá qué hacer con dicho dato e imprimirá en pantalla resultados erróneos. El tipo de dato asociado a una variable limita el conjunto de datos que puede almacenar, así como las operaciones aplicables sobre esa variable. Por lo tanto, una variable que pertenece a un tipo de dato *int* no podrá almacenar datos de tipo *char*; tampoco se podrán calcular operaciones propias de otros tipos de datos.

Las computadoras pueden trabajar con varios tipos de datos; los algoritmos y programas operan sobre éstos.

La asignación de tipos a los datos tiene dos objetivos principales:

1. Detectar errores de operaciones en programas.
2. Determinar cómo ejecutar las operaciones.

Los datos que utilizan los algoritmos y programas los podemos clasificar en *simples* o *compuestos*. Un *dato simple* es indivisible, no se puede descomponer. Un *dato compuesto* está integrado por varios datos. En este capítulo nos enfocaremos en los primeros.

Los tipos de datos simples son: numéricos (enteros y reales), lógicos (booleanos) y caracteres.

Tabla 2.1 Tipos básicos predefinidos

Tipo	Tamaño (bytes)	Rango
int (entero)	2	−32,768 a 32,767
float (flotante)	4	3.4 E−38 a 3.4 E+38
double (flotante de doble precisión)	8	1.7 E−308 a 1.7 E+308
char (carácter)	1	−128 a 127
void	0	sin valor

Los tipos de datos predefinidos son: numéricos, lógicos, caracteres y cadenas.

De ellos, tan sólo el tipo cadena es compuesto. Los demás son los tipos de datos simples considerados *estándares*. Esto quiere decir que la mayoría de los lenguajes de programación permiten trabajar con ellos. Por ejemplo, en lenguaje C es posible utilizar datos de tipo entero, real y carácter; sin embargo, los datos de tipo lógico sólo algunos compiladores los utilizan (por ejemplo, DEV-CPP).

El tamaño y el rango de algunos tipos de datos pueden variar, dependiendo del compilador utilizado. Algunas arquitecturas implementan tipos de datos de tamaño como lo podemos observar en la tabla 2.1; sin embargo en DEV-CPP el tipo *int* y *float* tienen 32 bits, el *char* 8 bits, y el *double* usualmente es de 64 bits; *bool* se implementa con 8 bits. En lenguaje C se puede utilizar el operador *sizeof* para determinar el tamaño de algunos tipos de datos en bytes (lo veremos más adelante, cuando veamos el tema de operadores).

Hay cinco datos básicos del lenguaje C: entero, coma flotante, coma flotante con doble precisión, carácter y sin valor.

2.2.1 Datos numéricos

Este tipo de datos se divide en enteros y reales.

Tipos enteros. Son aquellos números que no tienen fracciones o decimales. Pueden ser negativos o positivos y el rango es de −32,768 a 32,767, aunque este rango puede variar de un compilador a otro (por ejemplo DEV-CPP). Se almacenan internamente en 2 o 4 bytes de memoria y pueden ser: unsigned int, short int, int, unsigned long o long (cada uno de estos datos puede almacenar un rango diferente de valores). Cuando el rango de los tipos básicos no es suficientemente grande para sus necesidades, se consideran tipos enteros largos. Ambos tipos long requieren 4 bytes de memoria (32 bits) de almacenamiento.

Tipos reales o de coma flotante (float/double). Los tipos de datos flotantes contienen una coma (un punto) decimal, tal como 3.1416, pueden ser positivos y negativos formando el subconjunto de los números reales. Para representar números muy pequeños o muy grandes se emplea la notación de punto flotante, que es una generalización de la notación científica. En esta notación se considera al número real como mantisa y al exponente la potencia de 10 a la que se eleva este número, por ejemplo $340 = 3.4 \times 10^2$. C soporta tres formatos de coma flotante; el tipo *float* requiere 4 bytes de memoria, *double* 8 bytes, y *long double* 10 bytes.

2.2.2 Datos lógicos o booleanos

Hay lenguajes que sólo pueden tomar uno de dos valores: verdadero (*true*) o falso (*false*). En lenguaje C no existe el tipo lógico pero se puede implementar con un número entero, 0 es falso y cualquier número diferente de cero es verdadero. Algunos compiladores, como por ejemplo el DEV-C++ utilizan el *bool*.

2.2.3 Caracteres

El almacenamiento de caracteres en el interior de la computadora se hace en "palabras" de 8 bits (1 byte). Este tipo representa valores enteros en el rango −128 a +127. El lenguaje C proporciona el tipo *unsigned char* para representar valores de 0 a 255 y así representar todos los caracteres ASCII.

Una característica de la parte estándar del conjunto de caracteres (los 128 primeros) es que contiene las letras mayúsculas, las minúsculas y los dígitos, y que cada uno de estos tres subconjuntos está ordenado en su forma natural, por lo que podemos manejar rangos de caracteres bien definidos. Así, la siguiente expresión booleana decide si el carácter contenido en la variable *c* es una letra minúscula (*'a'<=c && c<='z'*). Los caracteres se almacenan internamente como números y por lo tanto se pueden realizar operaciones aritméticas con datos tipo *char*.

Existe también el dato tipo *cadena* (compuesto), que es una sucesión de caracteres que se encuentran delimitados por comillas; la longitud de una cadena es el número de caracteres comprendidos entre los delimitadores "[long_cad]".

Ejemplo



Pseudocódigo	Lenguaje C
caracter letra ← 'b', cadena [25]	`char letra = 'b', cadena [25];`
caracter car ← letra − 32	`char car = letra − 32;`

El valor de la variable car es B, ya que el código ASCII de b es 98 y al restarle 32 es igual a 66 (código ASCII de B).

2.2.4 Tipo void

Son datos vacíos o sin valor. Por ejemplo la función *main* no regresa valor alguno (nada): *void* main() o *void* main(void) porque tampoco tiene parámetros. Debemos tener cuidado, ya que esta característica es propia de algunos compiladores, pero por ejemplo en DEV-C++ el *main* tiene que regresar un entero (*int*), por el *return* 0, es decir *int* main(*void*) o *int* main(); pero las demás funciones predefinidas sí pueden utilizar el tipo *void*; esto lo analizaremos en detalle en el capítulo de funciones.

2.3 Variables



Una variable es un dato cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Es decir, representará un valor almacenado en memoria que se puede modificar en cualquier momento o conservar para ser usado tantas veces como se desee.

Hay diferentes tipos de variables: enteras, reales, caracteres y cadenas. Una variable que es de cierto tipo sólo puede tomar valores que correspondan a ese tipo. Si se intenta asignar un valor de tipo diferente se producirá un error.

El programador de lenguaje C es libre de denominar a sus variables con el nombre que considere más adecuado, siempre que se respeten las normas que mencionamos en la sección respectiva para nombrar un identificador. El lenguaje C acepta letras mayúsculas y minúsculas, sin embargo, son distintos los nombres en mayúsculas y minúsculas, es decir los nombres *lado* y *Lado* se refieren a variables diferentes.

Como se mencionó anteriormente, el uso de nombres largos no es recomendable ya que resultan más difíciles de teclear y además se utiliza más memoria para almacenar el nombre.

Los nombres de las variables nos deben indicar qué dato almacenan, de manera que resulte más fácil leer el programa. Así, la variable *nomAlumno* indica que almacena el nombre de un alumno.

2.3.1 Declaración de variables

Todas las variables deben ser declaradas antes de ser usadas. Cada variable por lo tanto tiene asociado un tipo, un *nombre* (identificador) y un *valor*. No se admiten como identificadores palabras reservadas del lenguaje de programación que se esté utilizando. Los nombres de variables que se elijan para el algoritmo o programa

deben ser significativos y tener relación con el objeto que representa. En lenguaje C la sintaxis para definir o declarar una variable es:

Pseudocódigo	Lenguaje C
tipo_dato ident_variable(s)	`tipo_dato ident_variable(s);`

Ejemplo




Pseudocódigo	Lenguaje C
entero i, j, k	`int i, j, k;`
real si	`float si;`
caracter s, nom[25]	`char s, nom[25];`

Las variables del mismo tipo pueden definirse con una definición múltiple, separándolas mediante " , " : int x, y, z;

Una variable puede declararse en cuatro lugares diferentes del algoritmo o programa:

- Fuera de todos los subprogramas o funciones (global).
- Dentro de un subprograma o función (local a la función).
- Dentro de un bloque enmarcado por llaves { } (local al bloque).
- Dentro de una instrucción, por ejemplo: for (int i=0; i<=10; i++).
- Como parámetro formal (local a la función).

2.3.2 Reserva de memoria

Cuando declaramos una variable le estamos diciendo al compilador que debe *reservar espacio en memoria*, que a cada espacio en memoria le asigne un nombre y un número determinado de bytes, dependiendo del tipo de dato asignado; también se le dice qué tipos de datos puede almacenar. En C una variable es una posición de memoria de la computadora con nombre (identificador), donde se almacena un valor con cierto tipo de dato. Una variable es un lugar donde se puede almacenar temporalmente un dato; las variables nos permiten guardar información.

Inicialización de variables

Las variables pueden también ser inicializadas (es decir, tomar un valor de inicio) en el momento de declararse:

Pseudocódigo	Lenguaje C
tipo_dato variable ← valor	`tipo_dato variable = valor;`

Ejemplo




Pseudocódigo	Lenguaje C
entero i ← 0	`int i = 0;`
real sal ← 30.5	`float sal = 30.5;`

2.4 Constantes

Una constante es un dato que permanece sin cambio durante el desarrollo del algoritmo o durante la ejecución del programa, es decir valores fijos que no pueden ser alterados por el usuario. La mayoría de los lenguajes de programación permiten el manejo de diferentes tipos de constantes; éstas pueden ser enteras, reales, caracteres y cadenas.

En la tabla 2.2 vemos que además de tener un valor, una constante también tiene un tipo de dato inherente (entero, real, carácter o cadena); el compilador C las evaluará en el momento de la compilación, en lugar de hacerlo en la ejecución.

Tabla 2.2 Tipos de constantes

Tipo de constante	Descripción y ejemplos
Enteras	Son una sucesión de dígitos precedidos o no por el signo + o − dentro de un rango determinado. Ejemplos: 234, −456, etcétera.
Reales	Son una sucesión de dígitos con punto adelante, al final o en medio y seguidos opcionalmente de un exponente. Ejemplos: 82.347, 0.63, 32.4e−05, 7.4e03
Carácter	Una constante carácter (*char*) es un carácter del código ASCII encerrado entre apóstrofos. Ejemplos: 'a', 'b', 'c'
Cadena	Una constante cadena es una secuencia de caracteres encerrados entre dobles comillas. Ejemplos: "123", "26 de noviembre de 1974", "Esto es una cadena"

En lenguaje C una constante se define por medio de la instrucción *#define* (directiva del procesador) o de la palabra *const*.

Pseudocódigo	Lenguaje C
constante iden_const ← valor	`#define iden_const valor` O `const tipo iden_const = valor;`

2.4.1 Uso de #define: creación de macros

El compilador C tiene un preprocesador incorporado. Si las líneas

#define LIMITE 100
#define PI 3.14159

se encuentran en un archivo que se está compilando, el preprocesador cambia primero todos los identificadores LIMITE por 100 y todos los PI por 3.14159, excepto los que estén en cadenas entre comillas. Una línea *#define* puede estar en cualquier lugar del programa, pero debe empezar en la columna 1 y sólo tendrá efecto en las líneas de archivo que le siguen. El preprocesador sólo cambiará los identificadores que se escriben con mayúsculas.

En una macro con argumentos, los argumentos se sustituyen en el texto de reemplazo:

```
//Ejemplo macros
#include<conio.h>
#include<stdio.h>
#define suma(n1,n2) n1+n2
#define multiplica(n1,n2) n1*n2
main()
{
    int x;
    x=suma(4,6);
    printf("Suma = %d\n",x);
    printf("Multiplicación = %d\n",multiplica(5,6));
    getch(); return 0;
}
```

Salida en pantalla, después de ejecutar el programa:
Suma = 10
Multiplicación = 30

2.4.2 Uso de const

El cualificador *const* permite dar nombres simbólicos a constantes. Su valor no puede ser modificado por el usuario.

Ejemplo

Pseudocódigo	Lenguaje C	
constante MAX ← 100	#define MAX 100	const int MAX = 100;
constante CAR ← 'a'	#define CAR 'a'	const char CAR = 'a';
constante CAR ← "a"	#define CAR "a"	const char CAR[4] = "a";
constante PI ← 3.1416	#define PI 3.1416	const float PI = 3.1416;
constante NOM ← "Marco"	#define NOM "Marco"	const char NOM[10] = "Marco";

Nota: "a" es diferente de 'a'.

La representación de 'a' como carácter y "a" como cadena en la memoria sería:

carácter   cadena
a   a\0

Las constantes se presentan en expresiones como:

Pseudocódigo	Lenguaje C
area ← PI * radio * radio	area = PI * radio * radio;

PI es una constante que ya tiene un valor igual a 3.1416.

2.5 Operadores

Un operador es un símbolo que permite relacionar dos datos en una expresión y evaluar el resultado de la operación.

Los programas de las computadoras se apoyan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad. Los operadores fundamentales son:

- Aritméticos.
- Relacionales.
- Lógicos.
- Asignación.

2.5.1 Operadores aritméticos

Los operadores aritméticos de la tabla 2.3 (+, −, *, /) pueden ser utilizados con tipos enteros o reales y sirven para realizar operaciones aritméticas básicas. Por ejemplo si a = 15 y b = 3, vemos los resultados con los diferentes operadores aritméticos.

Tabla 2.3 Operadores aritméticos

Operador			Ejemplo			
Pseudocódigo	Lenguaje C	Significado	Pseudocódigo	Lenguaje C	Operación	Resultado
+	+	Suma	a + b	a + b	Suma de a y b	18
−	−	Resta	a − b	a − b	Diferencia de a y b	12
*	*	Multiplicación	a * b	a * b	Producto de a por b	45
/	/	División	a / b	a / b	Cociente de a entre b	5
mod	%	Residuo	a mod b	a % b	Residuo de a entre b	0
^ ,**	pow	Potencia	a ^ b	pow(a, b)	a elevado a la b	3375

El operador % en lenguaje C, como se mencionó, calcula el residuo que queda al dividir dos números enteros; el siguiente programa muestra lo que realiza ese operador:

```
//Ejemplo %
#include <stdio.h>
#include <conio.h>
main(  )
{
    int x=−7,y=2  ;
    printf("%d",x%y);
    getch();
    return 0;
}
```

En la pantalla aparecerá:

−1

Ya que la división de −7/2 es 3 y el residuo es −1, debido a que el signo que se toma en cuenta es el negativo del numerador. Si x=7, y=2 el resultado hubiera sido +1.

2.5.2 Incremento y decremento

Estos operadores son propios de lenguaje C. En la tabla 2.4 podemos ver cómo funcionan.

Tabla 2.4 Operadores incrementales y decrementales

++	++ i	Se incrementa *i* en 1 y a continuación se utiliza el nuevo valor de *i* en la expresión en la cual esté i.
++	i ++	Utiliza el valor actual de *i* en la expresión en la cual esté *i*, y después se incrementa a en 1 (úsela y luego increméntela).
−−	−− i	Se decrementa *i* en 1 y a continuación se utiliza el nuevo valor de *i* en la expresión en la cual esté i.
−−	i −−	Se utiliza el valor actual de *i* en la expresión en la cual esté *i*, y después se decrementa a *i* en 1 (úsela y luego decreméntela).

Tanto ++ como −− pueden aplicarse a variables, pero no a constantes o a expresiones. Por ejemplo: ++i, i++ ambas son equivalentes a i=i+1. Veamos un ejemplo con este operador.

```
//Ejemplo incremento
#include <stdio.h>
#include <conio.h>
main()
{
    int a=7, b;
    b  =  ++a+3;  //a se incrementa en 1, le sumamos 3 y b recibe 11
    b  = b+a+++5;  //b recibe 11+8+5 y la a se incrementa
    printf ("%d,%d",a,b);
    getch();
    return 0;
}
```

En la pantalla aparecerá:

9,24

En el ejercicio anterior la variable *a* comienza con 7 y b = ++a+3, nos indica que primero incrementemos la variable *a* en 1 y luego la utilicemos, sumándole 3; por lo tanto *b* recibe 11. En la línea b=b+a+++5; la variable *b* recibe el valor de la *b* anterior + y el valor de *a*+5=11+8+5=24; en este caso primero utilizamos la variable *a* y luego la incrementamos en 1. Por último imprimimos el valor de ***a*=9** y el de ***b*=24**.

2.5.3 Operadores relacionales

Describen una relación entre dos valores; por lo tanto, se usan para expresar condiciones y comparar dos valores. El resultado de una expresión relacional es un valor tipo lógico o booleano, sólo puede ser *verdadero* o *falso*. El lenguaje C representa como verdadero el valor 1 y como falso el valor 0. En la tabla 2.5 se muestran los operadores relacionales y en la tabla 2.6 ejemplos de los mismos.

Tabla 2.5 Operadores relacionales

Operador		Significado
Pseudocódigo	**Lenguaje C**	
>	>	Mayor que
<	<	Menor que
=	==	Igual que
>=	>=	Mayor o igual que
<=	<=	Menor o igual que
< >	! =	Distinto a

Tabla 2.6 Ejemplos de operadores relacionales

Expresiones relacionales		Resultado	Valor binario
Pseudocódigo	**Lenguaje C**		
'A'<'B'	`'A' < 'B'`	V	1
'A'='a'	`'A' == 'a'`	F	0
8 <> 8.0	`8 != 8.0`	F	0
−124.2 < 0.003	`−124.2 < 0.003`	V	1
6.73 > 6.75	`6.73 > 6.75`	F	0
'A'<'a'	`'A' < 'a'`	V	1

En el último ejemplo el código ASCII de la A mayúscula es menor a la minúscula, ya que la primera tiene código 65 y la segunda 97.

Otro ejemplo: x+y<=3*z compara el valor de la suma de *x* y *y* con el triple del valor de *z*; si la suma es menor o igual que el triple de *z* entonces el resultado de la expresión es 1, y en el caso contrario es 0.

2.5.4 Operadores lógicos

Las expresiones lógicas pueden combinarse para formar expresiones más complejas utilizando los operadores lógicos. En la tabla 2.7 se muestran dichos operadores y en la tabla 2.8 ejemplos de los mismos. Estos operadores trabajan con operandos que son expresiones lógicas.

Tabla 2.7 Operadores lógicos

Pseudocódigo	Lenguaje C
y	&&
o	||
no	!

Tabla 2.8 Ejemplos de operadores lógicos

Expresiones lógicas o booleanas		Resultado	Valor binario
Pseudocódigo	**Lenguaje C**		
(5 > 3)	`(5 > 3)`	V	1
(8 < 4)	`(8 < 4)`	F	0
(5 > 3) y (8 < 4)	`(5 >3 && 8 < 4)`	F	0
(5 > 3) y (8 > 4)	`(5 >3 && 8 > 4)`	V	1
(5 > 3) o (8 < 4)	`(5 >3 || 8 < 4)`	V	1
(2 > 3) o (6 < 9)	`(2 >3 || 6 < 9)`	V	1
no (4 > 2)	`! (4 > 2)`	F	0
no (7 < 5)	`! (7 < 5)`	V	1

Como se muestra en la tabla anterior, se pueden evaluar una o más condiciones en una expresión, y siempre el resultado es un único valor lógico o booleano.

Para la evaluación de expresiones lógicas es importante conocer la tabla de verdad. En la tabla 2.9 hacemos referencia a los operadores lógicos de lenguaje C.

Tabla 2.9 Tabla de verdad

P	*Q*	*!Q*	*P && Q*	*P||Q*
verdad	verdad	falso	verdad	verdad
verdad	falso	verdad	falso	verdad
falso	verdad	falso	falso	verdad
falso	falso	verdad	falso	falso

Otros ejemplos en lenguaje C:

La expresión: a=!(4>1) almacena 0 en la variable a, debido a que la expresión 4>1 es verdadera y el operador *!* niega la expresión haciéndola falsa o igual a cero.

2.5.5 Operadores de asignación

El operador de asignación permite evaluar una expresión y asignar el resultado en una variable. Su sintaxis es:

Pseudocódigo	Lenguaje C
Identificador ← expresión	`Identificador = expresión;`

Con la asignación anterior le estaremos indicando a la computadora que evalúe la expresión y la almacene en la variable que define el identificador. Estos operadores permiten transferir el dato de una variable a otra. Así, la expresión x=a en lenguaje C transfiere el valor de *a* a la variable *x*. En la tabla 2.10 se muestran ejemplos de operadores de asignación.

Nunca debe escribirse la expresión a la izquierda del operador de asignación:

Pseudocódigo	Lenguaje C
(cal1 + cal2) ← promedio	`(cal1 + cal2) = promedio;`

Lo correcto es:

Pseudocódigo	Lenguaje C
promedio ← (cal1 + cal2)	`promedio = (cal1 + cal2);`

Tabla 2.10 Ejemplos de operadores de asignación

Pseudocódigo	Lenguaje C	Significado
c ← c + 21	`c = c + 21;`	Incrementa 21 a la variable c
x ← d − 12	`x = d - 12;`	Almacena en x la diferencia de d menos 12
j ← e * 5	`j = e * 5;`	Almacena en j el producto de e por 5
f ← f / 3	`f = f / 3;`	Divide el valor de f entre 3 y lo almacena en la variable f
g ← g mod 15	`g = g % 15;`	Divide el valor de g entre 15 y almacena el residuo en la variable g

Otros operadores de asignación en C

En la tabla 2.11 vemos otros operadores de asignación propios de lenguaje C: +=, *=, =, /= y %=, donde la sintaxis cuando se quiere utilizar uno de esos operadores es:

Identificador operador_asignación expresión;

Tabla 2.11 Otros operadores de asignación en C

Operador	Ejemplo	Equivalencia
+=	s += 10	s = s + 10
−=	r −= 3	r = r − 3
*=	m* = 4	m = m* 4
/=	d / = 2	d = d / 2
%=	x % = 33	x = x % 33
−=	x −= y*17	x = x − (y*17)

2.5.6 Operador condicional (?:)

Sustituye la sentencia **if... else**. El formato es:

<expresión booleana> ? <instrucción_sí> : <instrucción_no>

Por ejemplo: a = (b<2) ? 5 : 10;
es equivalente a: if (b<2) a=5;
else a=10;

la variable a toma el valor 5 cuando b<2, y el 10 en caso contrario.

Veamos otro ejemplo del operador condicional en el siguiente programa:

```
//Ejemplo Operador condicional
#include<stdio.h>
#include<conio.h>
main()
{
    int a=5, b=10, c;
    b<a ? printf("%d es el mayor",a): printf("%d es el mayor",b);
    getch(); return 0;
}
```

En la pantalla aparecerá:

10 es el mayor

La expresión booleana b<a es falsa por lo tanto se imprime 10 es el mayor.

2.6 Prioridad de los operadores



Al orden en que la computadora realiza las diferentes operaciones le llamamos *orden de prioridad*.

2.6.1 Prioridad de operadores aritméticos

1. *Paréntesis ( )*. Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro hacia afuera.
2. Dentro de una misma expresión o subexpresión, se evalúan en el siguiente orden:

   Potencia (^ o **)
   * , / ,mod
   +, −

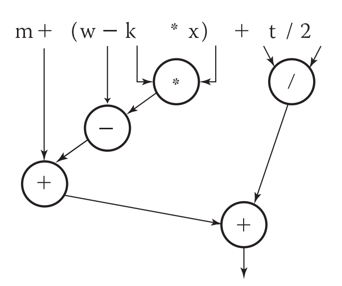
2.6.2 Regla asociativa

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de acuerdo con la tabla 2.12, en ella apreciamos la prioridad de todos los operadores en lenguaje C.

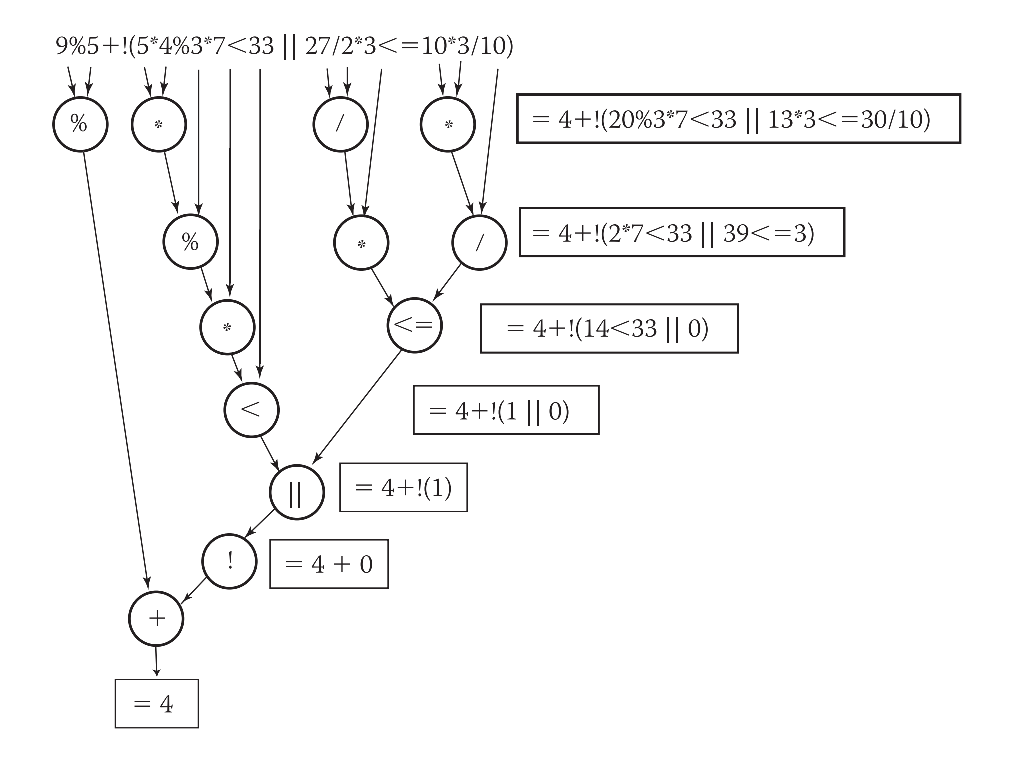
Tabla 2.12 Prioridad de los operadores en C

Categoría del operador	Operadores	Asociativa
Agrupación	( ), [ ]	Izquierda a derecha
Operadores monarios	−,++,−−,!, sizeof(tipo)	Derecha a izquierda
Multiplicación, división y residuo	*, /, %	Izquierda a derecha
Suma y sustracción aritmética	+, −	Izquierda a derecha
Operadores de relación	<, <=, >,>=	Izquierda a derecha
Operadores de igualdad	= = , !=	Izquierda a derecha
y	&&	Izquierda a derecha
o	||	Izquierda a derecha
Operador condicional	?:	Derecha a izquierda
Operadores de asignación	=, +=, −=, * =, =/=, %=	Derecha a izquierda

Los operadores de asignación tienen menor prioridad que todos los otros operadores. Por tanto las operaciones monarios, aritméticos, de relación, de igualdad y lógicos se realizan antes que las de asignación. El orden de las operaciones en pseudocódigo es:



Otro ejemplo en C:



2.7 Expresiones

Una expresión es el resultado de unir *operandos* mediante *operadores*. Los operandos pueden ser variables, constantes u otras expresiones; y los operadores, aritméticos, lógicos o relacionales. El resultado de una expresión es un dato numérico o un valor lógico. Para agrupar las expresiones utilizamos los paréntesis.

Según el tipo de datos que manipulan, se clasifican en:

- Aritméticas.
- Lógicas o booleanas.

Las expresiones lógicas o booleanas son expresiones relacionadas entre sí mediante operadores relacionales o lógicos. Una expresión lógica es una expresión que sólo puede tomar dos valores: verdadero o falso.

Ejemplos



Pseudocódigo	Lenguaje C
(num>=100)	`(num>=100)`
(num>=100) y (sexo = `M´)	`(num>=100 && sexo = 'M')`

Escritura de fórmulas matemáticas

Las fórmulas matemáticas se deben escribir en formato lineal. Esto obliga al uso frecuente de paréntesis que indiquen el orden de evaluación correcto de los operadores.

Ejemplos



Álgebra	Pseudocódigo	Lenguaje C
$w = \frac{(x + y + z)}{2}$	w ← (x + y + z)/2	`w = (x + y + z)/2;`
$x = \frac{5 - 3}{2 * 4}$	x ← (5 − 3)/(2*4)	`x =(5−3)/(2*4)`



2.8 Palabras reservadas

Son palabras propias del lenguaje, ya que el creador del mismo les dio un uso específico. No se pueden utilizar como identificadores. Las más utilizadas en pseudocódigo y lenguaje C las vemos en la tabla 2.13. En la tabla 2.14 aparecen todas las palabras reservadas de lenguaje C.

Tabla 2.13 Palabras reservadas en pseudocódigo y lenguaje C

Pseudocódigo	Lenguaje C	Función que realiza
abs()	`abs()`	Calcula el valor absoluto
caracter	`char`	Tipo de dato carácter
caso	`case`	Si se cumple un caso
caso contrario	`default`	Ninguna opción de la selectiva múltiple
define_tipo	`typedef`	Crea un nuevo nombre de tipo para un tipo de dato ya definido
desde	`for`	Estructura repetitiva (o de ciclo)
entero	`int`	Tipo de dato entero
fin	`}`	Fin del programa o de un bloque
hacer	`do`	Estructura repetitiva
imprimir	`printf`	Imprime en pantalla
imprimircad	`puts`	Imprime una cadena
inicio	`{`	Inicio del programa o de un bloque
leer	`scanf`	Lee una variable
leercad	`gets`	Lee una cadena de caracteres
limpiar_pantalla	`clrscr`	Borra el contenido de la pantalla
mientras	`while`	Estructura repetitiva
nada o ninguno	`void`	Valor nulo
principal	`main`	Programa principal
raizcuad	`sqrt`	Calcula raíz cuadrada
real	`float`	Tipo de dato real
registro	`struct`	Registro o estructura
regresa	`return`	Regresa valor a otra función
salir o interrumpir	`break`	Terminar el caso
segun_sea	`switch`	Estructura selectiva múltiple
si	`if`	Estructura selectiva
sino	`else`	La parte falsa de la selectiva

Toda palabra reservada en lenguaje C se escribe en minúscula, aunque algunas implementaciones como DEV-C++, puede tener excepciones (por ejemplo la función Sleep).

Tabla 2.14 Palabras reservadas de ANSI C

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	tamof	static
struct	switch	typedef	union
unsigned	void	volatile	while

2.9 Comentarios

Los comentarios son útiles para identificar los elementos principales de un programa o explicar la lógica derivada de éstos. Deben ser breves y se deben evitar ambigüedades.

Los comentarios en cualquier lenguaje de programación sirven para que el código fuente sea más entendible, aumentan la claridad de un programa, ayudan a la documentación y bien utilizados nos pueden ahorrar mucho tiempo.

Se deben utilizar comentarios sólo cuando sean necesarios, por ejemplo:

- Al principio del programa: nombre del programa, autor o autores, fecha de elaboración, etcétera.
- En la declaración de variables y constantes cuyo identificador no sea suficiente para comprender su uso.
- En los cierres de bloques con '}', para indicar a qué sentencias de control de flujo pertenecen, principalmente cuando existe mucho anidamiento de sentencias y/o los bloques contienen muchas líneas de código.

Los comentarios los reconocerá la persona que elaboró el programa o cualquier otro programador, inclusive después de un tiempo. Para el compilador, los comentarios son inexistentes, por lo que no generan líneas de código, lo que permite abundar en ellos tanto como se desee, aunque con medida.

En el lenguaje C se toma como comentario todo carácter interno a los símbolos: /* */ o iniciándolos con //. Los comentarios pueden ocupar uno o más renglones.

Ejemplo

```
/*  Este es un Comentario      */          C estándar
//  Este es un Comentario                  C++
```

2.10 Conversión de tipos

Cuando se evalúa una expresión aritmética que contiene diferentes tipos de datos, el compilador convierte todos ellos a un tipo único. Estas conversiones pueden aumentar o disminuir la precisión del tipo al que se convierten los elementos de la expresión. La conversión se hace operación a operación y se lleva a cabo según el tipo que esté más a la derecha de la siguiente relación:

char → *int* → *unsigned int* → *long* → *unsigned long* → *float* → *double* → *long double*

Ejemplo



```
char c;
int i;
long l;
float f;
double d;
x = ( i * c  % l) − ( d / f ) + f;
```

Tomando en cuenta el orden de prioridad, primero evaluamos el contenido de los paréntesis, por lo tanto en los primeros paréntesis evaluamos los tipos de izquierda a derecha:

i * c   es un *int*, ya que el tipo int está a la derecha del tipo char en nuestra relación.
i % l   es un *long*, el tipo long está a la derecha del tipo int en nuestra relación.
d / f   es un *double*, ya que el tipo double está a la derecha del tipo float.

Evaluamos de nuevo de izquierda a derecha y queda de la siguiente forma:

long − double + float = double + float = double

Para comprender lo anterior ejecute el siguiente programa:

```
#include < stdio.h >
main (){
```

(continúa)

(*continuación*)

```
    printf("division entera: 5/2 es % 7d\n", 5/2);
    printf("division flotante: 5./2. es %7.2f\n", 5./2.);
    printf("division mixta: 5./2 es %7.2f\n", 5./2);
}
```

El resultado de la primera división es 2, ya que los dos valores son enteros; en las dos últimas divisiones se obtendrá 2.50 ya que es suficiente que un valor sea real (numerador o numerador).

2.10.1 Conversión forzada de tipos de datos

Además de la conversión automática, el lenguaje C ofrece la posibilidad de forzar la conversión de un tipo de datos en otro tipo diferente. Esta conversión forzada es conocida como "casts":

(tipo) expresión

Ejemplo



```
int a = 3, b = 2;
float c, d;
c = a / b;    //el tipo de c es int, ya que a y b son tipo int; el resultado es 1
d = (float)a / b; // el valor de d es 1.5
```

2.10.2 El operador sizeof

sizeof proporciona la cantidad de memoria ocupada por los elementos de datos, sizeof indica el número de bytes utilizado por cualquier variable particular en un determinado compilador.

```
# include <stdio.h>
# include <conio.h>
main() {
    printf ("Tipo char: %d bytes\n", sizeof(char));
    printf ("Tipo short: %d bytes\n", sizeof(short));
    printf ("Tipo int: %d bytes\n", sizeof(int));
    printf ("Tipo long: %d bytes\n", sizeof(long));
    printf ("Tipo float: %d bytes\n", sizeof(float));
    printf ("Tipo double: %d bytes\n", sizeof(double)); getch(); return 0;
}
```

Ejercicios complementarios de los datos y operaciones básicas en pseudocódigo

Ejercicio 1. Realice un algoritmo para indicar las actividades que efectúa el día martes.

1. ______________________________
2. ______________________________
3. ______________________________
4. ______________________________
5. ______________________________


6. ____
7. ____
8. ____
9. ____

Ejercicio 2. Escriba válido o inválido en cada identificador para el pseudocódigo y responda ¿por qué?

Identificador	Válido o inválido	¿Por qué?
area		
área		
dato 1		
dato_1		
1radio		
radio1		

Ejercicio 3. Añada válido o inválido a cada declaración de variable en pseudocódigo.

Declaración de variable	Válido o inválido
entero a	
entero _a	
real x	
real x1	
real 1x	
caracter %s	

Ejercicio 4. Escriba válido o inválido a cada declaración de constante en pseudocódigo. Si la constante es válida, especifique el tipo respectivo.

Declaración de constante	Válido o inválido	Tipo de constante
constante MAX ← 20		
constante X ← 7.8E+0.3		
constante C ← 'x'		
constante ← 3.1415		
constante N ← "Marco"		
constante CAR ← c		
constante G ← 9.8		
constante NOM ← Lucy		
constante X ← 9.3e-2		
constante W ← 'abc'		

Ejercicio 5. Calcule las siguientes expresiones en pseudocódigo y escriba el resultado respectivo.

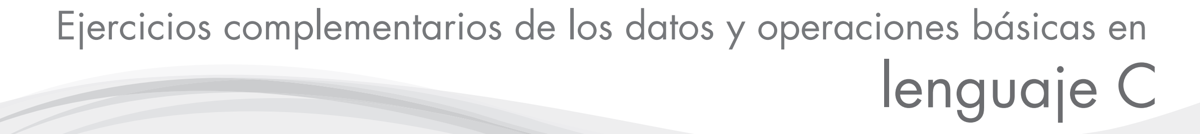
1. 9+3*16/ 2**3−5/2 =

2. 15/3+4*3^2−5*6 =

3. (6*3+8−9/3+2^4)/2 =

Ejercicio 6. Complete la columna de resultado y de valor binario según la expresión.

Expresiones lógicas	Resultado	Valor binario
(2 > 3)		
(8 < 15)		
(7 > 5) y (4 < 9)		
(12 > 6) y (13 < 9)		
(12 > 6) o (13 < 9)		
no (2 > 4)		
no (5 < 8)		



Ejercicios complementarios de los datos y operaciones básicas en lenguaje C

***a*) Responda las siguientes preguntas:**

1. ¿Qué es un identificador?

2. ¿Cuáles son las reglas para nombrar un identificador?

3. ¿Qué es un tipo de dato? Mencione tres tipos de datos en el lenguaje C y sus características.

4. ¿Qué es una variable?

5. ¿Qué es una constante?

6. ¿Qué es una expresión?
___
7. ¿Qué es un operador?
___
8. Mencione tres tipos de operadores de lenguaje C.
___
9. ¿Qué es el orden de prioridad?
___
10. ¿Cuál es el orden de prioridad de los operadores aritméticos?
___
11. ¿Cuándo se deben utilizar los paréntesis en una expresión? Mencione un ejemplo.
___
12. ¿Qué diferencia existe entre los operadores = y == en lenguaje C?
___
13. ¿Qué es una palabra reservada?
___
14. ¿Qué es un comentario y cómo se representa en lenguaje C?
___

***b*) Investigue:**

1. Una breve reseña histórica del lenguaje C.
2. Cómo obtener los rangos de valores de cada tipo, auxíliese de las librerías: limits.h, para los de tipo entero y float.h para los de tipo real.

Ejercicio 1. Tache los identificadores que no son válidos para lenguaje C.

1. _numero
2. número
3. año
4. home
5. dinero$
6. base_1
7. 2variables
8. el punto
9. scanf
10. end

Ejercicio 2. Escriba las siguientes expresiones matemáticas como expresiones de C.

1. $\frac{x+y}{x-3}+2x=$
2. $\frac{x+y}{x/5-3/y}+2x=$
3. $\frac{2+5y}{x-3}+4x-\sqrt[3]{7-y}=$
4. $\frac{2xy+5y}{x-3}-\frac{4x-xy}{\sqrt[3]{7-y/(4+x)}}=$

Ejercicio 3. Calcule las siguientes expresiones en lenguaje C y escriba el resultado respectivo.

1. 3+2*(k=7/2)=
2. 9+pow (3, 2) / (9−11) =
3. –3*7/2+pow(2,3)/4–2=
4. 6+2*(5–(4+2))–3*(7.2+9.0*6)=

5. 8+7% 5% 3 * 2 =
6. −7% − 5 * 9 / 4 =
7. !(17>=pow(6,2)||(13−7*3%2!=4*5%2))=
8. 7−((21>=4*pow(2,3) && 9 > 8 && 22 > 5)||!(5*2 < 3 + 16*32−pow(3,2)))*3 =
9. 2+7*(6*(1−(7+2)/9))=
10. 6>5||16==15 && !(9>7)=
11. 4*((!(7/4>2)||9>9) && (3>=7/3||1+4>=9/2))+5=

Ejercicio 4. Considerando las variables: int i=7, j=4, k; float f; char c, realice la corrida a mano de las siguientes expresiones, escribiendo a la derecha los valores (siga el orden establecido).

1. i+= j++;	i =	j =
2. j+= ++i+7;	j =	i =
3. k = (j++<i) ? i−2 : j*2;	k =	j =
4. j * = i+k;	j =	
5. j / = i−3;	j =	
6. j = i ++−5;	j =	i =
7. j += i%(k−1);	j =	
8. f = (float) i/3;	f =	
9. k = j−'7'+2;	k =	
10. c = 2* j + 1/2;	c =	
11. k = 3! = 7;	k =	
12. k = j = 5;	k =	j =
13. i + j = k;	k =	





Programación estructurada

Al término de este capítulo, el alumno será capaz de

- Identificar y utilizar técnicas para la formulación de algoritmos (pseudocódigo y diagramas de flujo), y su codificación en un lenguaje de alto nivel, utilizando las diferentes estructuras de control en un programa.

Contenido



3.1 Definición

La programación estructurada es un paradigma o forma de programar. Es un conjunto de técnicas que nos permiten desarrollar programas fáciles de escribir, verificar, leer y mantener e incluyen:

1. Diseño descendente (*top-down*).
2. Estructuras de datos.
3. Estructuras de control.
4. Programación modular.

En este capítulo nos enfocaremos en el tercer punto.

Diseño descendente (*top-down*)

En la programación estructurada las instrucciones están ordenadas u organizadas de arriba a abajo, lo que facilita el diseño del algoritmo, el entendimiento del código y por consiguiente el mantenimiento del mismo.

Estructuras de datos

Son un conjunto de datos donde podemos almacenar y acceder a elementos individuales de datos, por lo que pueden separarse en los elementos que la forman.

Programación modular

Otra característica que tiene la programación estructurada es que el problema se puede dividir en secciones o partes (módulos). Este tipo de programación permite resolverlo de manera más sencilla y en forma paralela si es necesario, es decir por un equipo de personas.



3.2 Estructuras de control y su clasificación

Estas estructuras controlan cómo se ejecutan los programas, es decir el orden de las instrucciones, ya que tienen un solo punto de entrada y un punto de salida. En la programación estructurada se mezclan las estructuras de control y las podemos clasificar en:

Estructuras de control	Secuencial. Selectiva. Repetitiva o de iteración condicionada.

Estructura de control secuencial

Las instrucciones se ejecutan en orden, una por una desde la primera hasta la última, es decir el programa ejecuta todas las instrucciones del programa en el orden establecido sin saltarse ninguna de ellas.

Estructura de control selectiva o alternativa

De acuerdo con una condición que puede ser verdadera o falsa se elige una opción, la cual realiza una acción (una o varias instrucciones). La condición puede ser simple o compuesta (una o varias).

Estructura de control repetitiva o de iteración condicionada

Una acción se repite una cantidad definida o indefinida de veces mientras una condición sea verdadera.

La lógica de programación se centra sobre todo en el cuerpo del programa, utilizando las estructuras de datos y las de control, además de la programación modular. Para diseñar los programas de computadora,

comúnmente se utilizan diferentes estructuras de control a fin de poder llegar a la solución de un problema, cuáles y cuántas dependerán del problema mismo.

Para facilitar la comprensión de la lógica de programación, primero realizaremos el diagrama de flujo, el pseudocódigo y lenguaje C de algunos ejercicios y después solamente el pseudocódigo y lenguaje C.

3.3 Estructura de un algoritmo y de un programa

En la programación estructurada se recomienda tener un orden, tanto en los algoritmos, como en los programas; por ejemplo, la estructura de un algoritmo en pseudocódigo y de un programa lenguaje C es la siguiente:

Pseudocódigo	Lenguaje C
	`Comentarios`
	`Declaración de archivos de cabecera (librerías)`
Prototipo de las funciones	`Prototipo de la funciones`
Declaración de variables globales y constantes	`Declaración de variables globales y constantes`
Programa principal ( )	`main() // Programa principal`
inicio	`{`
Declaración de constantes (locales)	`Declaración de constantes (locales)`
Declaración de variables (locales)	`Declaración de variables (locales)`
Cuerpo del programa (estructuras de control)	`Cuerpo del programa (estructuras de control)`
fin	`}`
Declaración de funciones	`Declaración de funciones`
inicio	`{`
fin	`}`

Programa principal ()[1] en pseudocódigo es el equivalente en lenguaje C a la función main().

A continuación explicaremos cada parte de la estructura de un programa en lenguaje C.

3.3.1 Comentarios

La primera línea de un programa en C suele ser una línea de comentario, en la cual se escribe el nombre del programa, lo que realiza, los datos del autor, la fecha y la versión. Las líneas de comentario en C estándar se encierran entre los caracteres /* y */. En C++ los comentarios se inician con //.

3.3.2 Declaración archivos de cabecera o encabezado (librerías o bibliotecas)

Indican al compilador que en esta posición se incluyan las líneas de sentencias que están dentro del archivo que se declara; son archivos estándar proporcionados por el fabricante del compilador, y se suelen declarar funciones, variables y constantes que van a ser utilizadas por las sentencias que el programador va a manejar en las siguientes líneas del programa.

Para llamar un archivo de inclusión o cabecera es necesario hacer uso de la directiva *#include*, la cual tiene la siguiente *sintaxis*:

#include nombre_archivo_cabecera

[1] De aquí en adelante sólo utilizaremos la palabra reservada principal.

Donde nombre_archivo_cabecera es el nombre de un archivo que contiene las declaraciones y definiciones para un propósito específico. Este archivo debe tener la extensión .h.

Ejemplos



```
#include "stdio.h"
#include <stdio.h>
```

Las comillas le dicen a C que busque primero en el directorio de trabajo actual el archivo de inclusión; si no lo encuentra, busca entonces en el directorio especificado en la línea de órdenes, y finalmente si aún no lo ha encontrado entonces busca en el directorio estándar que se haya definido durante la instalación.

Si el archivo de inclusión se encierra entre signos menor y mayor que, el compilador busca primero en el directorio especificado en la línea de órdenes; si no lo encuentra busca entonces en el directorio estándar. Jamás busca en el directorio de trabajo.

Si en la inclusión se especifica nombre de ruta o camino, entonces busca en dicho directorio.

Ejemplos



```
#include "c:\cabecera\stdio.h"
#include <c:\include\ctype.h>
```

Bibliotecas o librerías de cabecera más utilizadas en C

stdio.h contiene declaraciones de rutinas de entrada/salida.
math.h contiene declaraciones de funciones matemáticas.
string.h contiene funciones con cadenas.
conio.h consola y puertos de E/S.
ctype.h clasificador de caracteres.

3.3.3 Prototipos de funciones

Un prototipo es la declaración de una función que sólo contiene la cabecera; a diferencia de la declaración completa, al final lleva punto y coma. El prototipo le avisa al compilador que existe una función que va a regresar un determinado tipo de dato y qué parámetros utilizará.

3.3.4 Declaración de variables globales y constantes

Las constantes y variables son elementos que tienen el objetivo de identificar los datos que se utilizan en las operaciones y cálculos que realiza el programa y que se almacenan en la memoria de la computadora. Si las variables globales son declaradas en esta parte, pueden ser utilizadas en todo el programa.

3.3.5 El programa principal o función principal main()

El programa principal contiene el flujo del programa llamando a las funciones necesarias para su funcionamiento. La función *main()* indica dónde empieza el programa, cuyo cuerpo principal es un conjunto de instrucciones delimitadas por dos llaves, una inmediatamente después de la declaración *main()*, " { ", y otra que finaliza el listado, " } ". Todos los programas C arrancan del mismo punto: la primera instrucción dentro de dicha función.

3.3.6 Declaración de funciones

Se declaran las funciones que utiliza el programa y que no están definidas en las librerías. Las funciones son un conjunto de sentencias que realizan una tarea concreta y que retornan un dato como resultado de las operaciones que han realizado. Es importante recordar que lo mínimo que debe contener un programa en lenguaje C es un archivo de cabecera y el main(). Por ejemplo:

Ejemplos



```
#include <stdio.h>
main()
{
    printf("Bienvenido al Lenguaje C");
}
```

Ejemplo de estructura de un programa típico en lenguaje C

Ejemplos



Comentario del inicio del programa

```
/* Programa que muestra la estructura de un programa en C*/
```

Declaración de archivos de cabecera (librerías)

```
#include <stdio.h>
#include <conio.h>
```

Prototipo de las funciones:

```
void bienvenida (char nombre);
```

Declaración de variables globales y constantes:

```
char nombre[30];
```

Programa principal:

```
main()
{
    printf("¿Cuál es tu nombre:?\n");
    scanf("%s",&nombre);
    bienvenida(nombre);   //Llamada a función
    getch();   return 0;
}
```

Declaración de funciones:

```
void bienvenida (char nomb[])
{
    printf("\nhola %s Bienvenido(a) al Lenguaje C", nomb);
    getch();
}
```

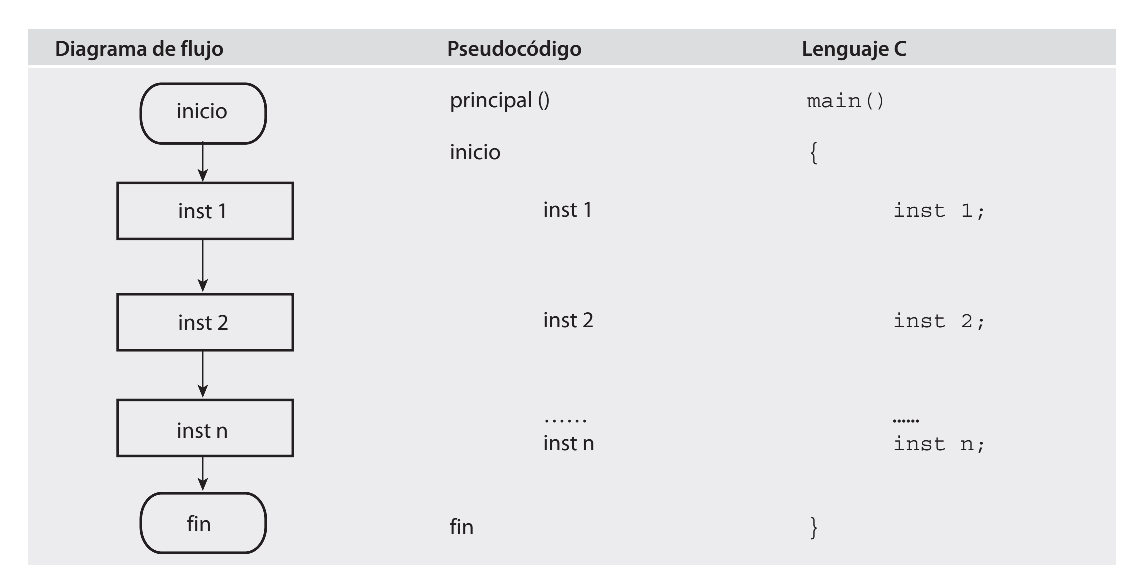
El programa completo queda de la siguiente manera:

```
#include <stdio.h>
#include <conio.h>
void bienvenida (char nomb[]);
char nombre[30];
main()
{
   printf("¿Cuál es tu nombre:?\n");
   scanf("%s",&nombre);
   bienvenida  (nombre);
}
void bienvenida  (char nomb[])
{
   printf("\nhola  %s Bienvenido(a) al Lenguaje C", nomb);
   getch();
}
```

3.4 Estructura de control secuencial

Es la estructura más sencilla ya que el programador identifica los datos de entrada, los procesa y muestra o imprime los datos de salida.

La estructura secuencial se puede representar de la siguiente forma:

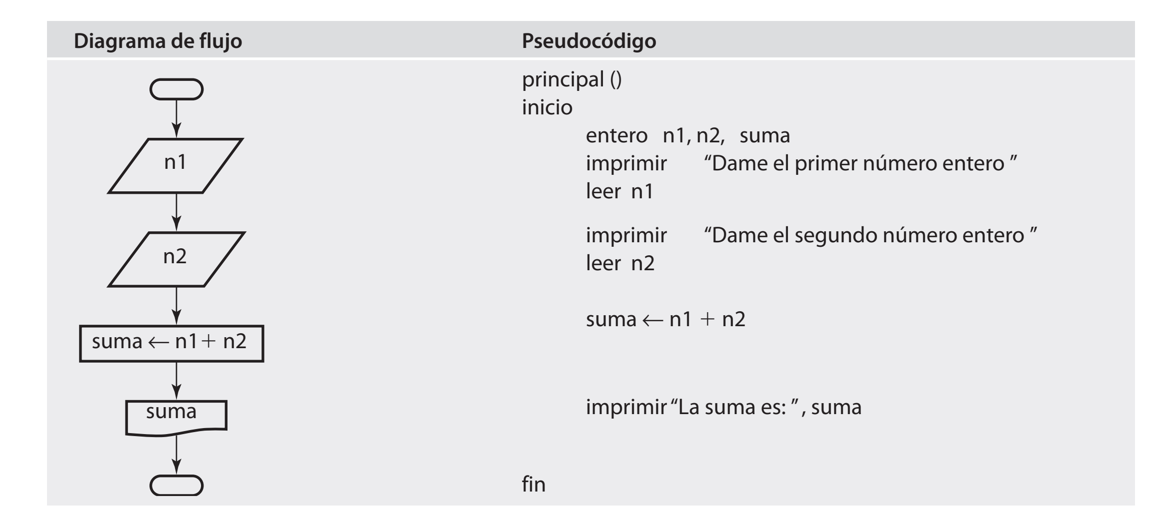


A continuación presentamos los ejercicios resueltos; para tal efecto debemos retomar lo aprendido en secciones anteriores respecto a la *asignación*. Además debemos distinguir la *entrada* y la *salida* de datos:

Significado	Diagrama de flujo	Pseudocódigo	Lenguaje C
Entrada		leer leercad	`scanf` `gets`
Salida		imprimir imprimircad	`printf` `puts`

3.4.1 Ejercicios resueltos de la estructura de control secuencial

Ejercicio 1. Sume dos números enteros.



Para la parte lógica de la programación nos centraremos en las tres primeras etapas del desarrollo de un programa.

1. *Entender el problema*.

   En este caso sumar dos números; el término suma es claro y conocido.

2. *Análisis del problema.*
   Al realizar un análisis nos arroja los siguientes datos:

   Datos de entrada : n1, n2
   Datos de salida : suma
   Fórmula : suma = n1 + n2

3. *Diseño del algoritmo*.

   Se desarrollaron el pseudocódigo y el diagrama de flujo arriba descritos.

Explicación del diseño:

Como se aprecia en la tabla 3.1, el nombre de las variables las elige usted; se recomienda que hagan referencia a lo solicitado o calculado.

Tabla 3.1 Nombre y descripción de las variables

Nombre de la variable	Descripción
n1	Primer número o número 1 a introducir
n2	Segundo número o número 2 a introducir
suma	Suma de los dos números introducidos

Para este programa se declaran tres variables de tipo entero, dos variables que son los datos de entrada que no se conocen y se necesitan leer o pedir al usuario (*n1* y *n2*) y una variable (dato de salida) en la cual se va a guardar el resultado (*suma*). Cabe hacer notar que en el diagrama de flujo no se acostumbra declarar ni las variables ni las constantes.

La instrucción *imprimir* muestra en pantalla todo lo que se encuentra entre las comillas, y en la parte que no tiene comillas se muestra el valor de la variable respectiva. Por lo tanto, los letreros los distinguimos de las variables por las comillas y los separamos con comas, si se requiere imprimir o mostrar el valor de varias variables. Cabe hacer notar que en un *diagrama de flujo* los letreros previos a la lectura de un dato o impresión de una variable son opcionales, en el presente ejercicio no aparecen.

La instrucción *leer* solicita los datos al usuario, guardando éstos en las variables *n1* y *n2*. La expresión suma ← *n1* + *n2* primero realiza la suma de los valores almacenados a las variables *n1* y *n2* y el resultado lo guarda en la variable *suma*, la cual se muestra en pantalla en la última instrucción.

Ejemplos

si n1 = 6 y n2 = 9 se mostrará: La *suma es*: 15.

Nota 1: Solamente se muestra el valor de la variable suma, mas no el nombre de la variable.

Nota 2: Todo programa solicita los datos que desconoce o que necesita saber, los almacena en variable(s) y luego utiliza el valor de la variable para realizar los cálculos necesarios según sea el resultado que desea obtener, almacenando este resultado en alguna variable y luego mostrando el valor de la variable o imprimiendo directamente el resultado, como en el ejercicio siguiente.

De acuerdo con lo que acabamos de ver en el ejercicio, a continuación describimos un algoritmo en pseudocódigo siempre que tengamos fórmulas:

1. principal()
2. inicio
3. Declaración de variables de entrada y salida.
4. Leer datos de entrada, los encontramos a la derecha del operador de asignación.

5. Procesar fórmula.
6. Imprimir datos de salida, los encontramos a la izquierda del operador de asignación; por lo tanto por cada fórmula sólo abra un dato de salida.
7. fin.

El algoritmo presentado es para una fórmula, pero también se puede implementar para varias fórmulas, respetando los pasos 1, 2 y 7. Se pueden repetir los pasos 4, 5 y 6 en secuencia para cada fórmula o aplicando cada paso para todas las fórmulas, por ejemplo si se tienen tres fórmulas en el paso 6 se imprimirían los 3 resultados.

4. *Código en lenguaje C*

Ahora veamos el significado de cada una de las líneas del programa.

Lenguaje C

```
   /* Ejercicio  1. Sume dos números enteros  */

#include  <stdio.h>
#include  <conio.h>
main()
{
   int   n1, n2, suma;
   clrscr();
   printf("Dame el primer número");
   scanf("%d",&n1);
   printf("Dame el segundo número");
   scanf("%d",&n2);
   suma  = n1  + n2;
   printf("La suma es  :  \n  %d",suma);
   getch();
   return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame el primer número 10

Dame el segundo número 30

La suma es :

40

Ejercicio 1. Sume dos números enteros.

Es un comentario. El compilador de Turbo C ignora todo lo que está entre los símbolos de comienzo (/*) y fin (*/) de un comentario. Los comentarios delimitados por estos símbolos pueden ocupar varias líneas. Cuando se utilice el símbolo //, se tendrá que escribir al principio de cada renglón, por lo que es más conveniente el símbolo anterior para varias líneas.

#include <stdio.h>

Le dice a Turbo C que en el proceso de compilación incluya un archivo denominado *stdio.h*. La sentencia *#include* no es una instrucción C. El símbolo # la identifica como una directiva, es decir, una orden para el preprocesador de C, responsable de realizar ciertas tareas previas a la compilación. Los archivos **.h* se denominan *archivos de cabecera*. Todos los programas C requieren la inclusión de uno o varios archivos de este tipo, por lo que normalmente es necesario utilizar varias líneas *#include*. El archivo *stdio.h* contiene las declaraciones de las funciones de entrada/salida, así como definiciones de constantes simbólicas y algunas otras definiciones de utilidad.

#include <conio.h>

Este archivo contiene funciones como son clrscr()[2] y getch().

main ()

[2] La instrucción clrscr() no viene incluida en librería conio.h de DEV-C++, para poderla utilizar se necesita instalar el conio2.h, o en su defecto no usarla.

Es el nombre de una función. Un programa C se compone de una o más funciones, pero al menos una de ellas debe llamarse *main()* (*principal*), pues los programas C empiezan a ejecutarse por esta función. La palabra *void* es opcional; indica que la función *main()* no tiene valor de retorno ni argumentos. Podríamos haber escrito void[3] main() y no hubiéramos necesitado la instrucción *return 0*.

Los paréntesis identifican a *main()* como una función. Generalmente, dentro de ellos se incluye información que se envía a la función. En este caso no hay traspaso de información, por lo que no hay nada escrito en su interior. Aun así son obligatorios. El *cuerpo de una función* (conjunto de sentencias que la componen) va enmarcado entre llaves { y }. Ése es el significado de las llaves que aparecen en el ejemplo (*inicio* y *fin* en pseudocódigo).

```
int  n1, n2, suma;
```

Es una sentencia declarativa; indica que se van a utilizar tres variable *n1, n2*, *suma* que son de tipo entero. La instrucción *int* es una palabra clave de C que identifica uno de los tipos básicos de datos. En C es obligatorio declarar todas las variables antes de ser utilizadas. El ";" identifica la línea como una sentencia C.

```
clrscr();
```

Esta función permite borrar pantalla (*clear screen*) y pertenece a la librería <conio.h>.

```
printf("Dame el primer Número");
```

La función *printf()* pertenece a la biblioteca estándar de C y se encuentra en el archivo *stdio.h*, de ahí que sea necesaria la sentencia *#include <stdio.h>*. Esta función es el equivalente de *imprimir* en pseudocódigo.

```
scanf("%d";&n1);
```

La sentencia consta de dos partes:

- El nombre de la función: *scanf()*.
- Los argumentos. Son los valores que aparecen entre los paréntesis separados por una coma, en este caso hay dos:
  - "%d".
  - &n1.

scanf() es una función de la biblioteca estándar de C (como *printf()*), que permite leer datos del teclado y almacenarlos en una variable. En el ejemplo, el primer argumento siempre va dentro de las comillas: %d, le dice a *scanf()* que tome del teclado un número entero. El segundo argumento, *&n1*, indica en qué variable se almacenará el dato leído. El símbolo *&* antes del nombre de la variable es necesario para *scanf()* ya que nos indica la dirección de la memoria en donde ubicaremos la variable en cuestión. Esta función es el equivalente de *leer* en pseudocódigo.

```
printf("Dame el segundo Número");  scanf("%d";&n2);
```

Es igual al proceso que utilizamos para leer n1 en las líneas anteriores.

```
suma = n1 + n2;
```

Se almacena en la variable *suma* el resultado de sumar la variable *n1* más *n2*.

```
printf("La suma es : \n %d",suma);
```

Como vemos a diferencia del imprimir en pseudocódigo la variable necesita el formato respectivo. El primer argumento es una *cadena de formato*. En esta cadena el texto aparecerá idéntico en pantalla. En la cadena de formato pueden aparecer *códigos de formato* y *caracteres de escape*.

Un *código de formato* comienza por el símbolo % e indica la posición dentro de la cadena en donde se imprimirá el segundo argumento, en este caso, la variable *suma*. En este ejemplo, *%d* indica que en su lugar se visualizará un número entero decimal.

[3] En DEV-C++ no es permitido utilizar void main(), sólo acepta int main() o simplemente main().

Un *carácter de escape* comienza por el símbolo \. La secuencia *n* es el carácter *nueva línea* y equivale a la secuencia LF + CR (salto de línea + retorno de cursor).

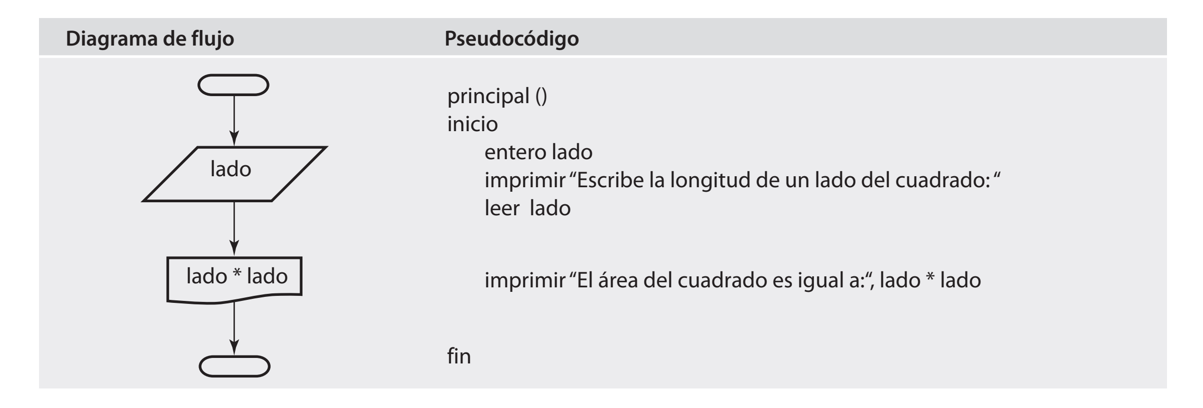
getch();

Esta función hace una pausa, pues espera a que oprima cualquier tecla y con ello puede ver el resultado antes de que regrese al editor del compilador. Pertenece a la librería <conio.h>.

return 0;

Cualquier función en C debe regresar algún valor, exceptuando las funciones que regresan un tipo *void* (nada); en nuestro ejemplo sólo utilizamos la función *main()* y por lo tanto por *default* al no aparecer un tipo de retorno el tipo es entero *(int)*, y por lo tanto como la función *main*() no regresa algún valor le indicamos que regrese o retorne 0 (cero), que es nada. También es permitido escribir *return (0).*

Ejercicio 2. Área de un cuadrado.



Pseudocódigo

```
principal ()
inicio
    entero lado
    imprimir "Escribe la longitud de un lado del cuadrado: "
    leer  lado

    imprimir "El área del cuadrado es igual a:", lado * lado

fin
```

Este programa declara solamente una variable *lado*, la cual nos sirve para guardar el valor del lado de un cuadrado, pero como verán no existe ninguna variable para almacenar el resultado (área), así que el cálculo lo haremos directamente al momento de imprimir el resultado; primero se hace el producto del *lado * lado*, y el resultado se imprimirá después de la etiqueta o mensaje.

Ejemplos



Si lado es igual a 5 el programa imprimirá: *El área del cuadrado es igual a: 25.*

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
main()
{
   int lado;
   char unid[12];
   clrscr();
   printf("Dame las unidades:  ");
   scanf("%s",&unid);
   printf("Escribe la longitud de un lado del cuadrado:  ");
   scanf("%i",&lado);
   printf("El área del cuadrado es igual a: %i  %s",lado*lado, unid);
   getch();
   return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame las unidades: centímetros

Escribe la longitud de un lado del cuadrado: 10

El área del cuadrado es igual a: 100 centímetros

La diferencia entre el pseudocódigo y el código en C, es que agregamos una variable cadena para las unidades. La variable *unid* es de tipo *char [12]* (cadena de 12 caracteres), por lo tanto el formato para leer e imprimir dicha variable es *%s*.

Nota: Algunos compiladores de lenguaje C no aceptan para cadenas el formato %s, por lo que se debe utilizar la función *gets(),* la cual veremos posteriormente.

En el último *printf* imprimimos dos valores: una expresión (lado*lado) y una cadena.
Qué sucede si la longitud de un lado del cuadrado es: 200.
Dame las unidades: metros.
Escribe la longitud de un lado del cuadrado: 200.
El área del cuadrado es igual a: −25536 metros.

¿Por qué se imprime este resultado?

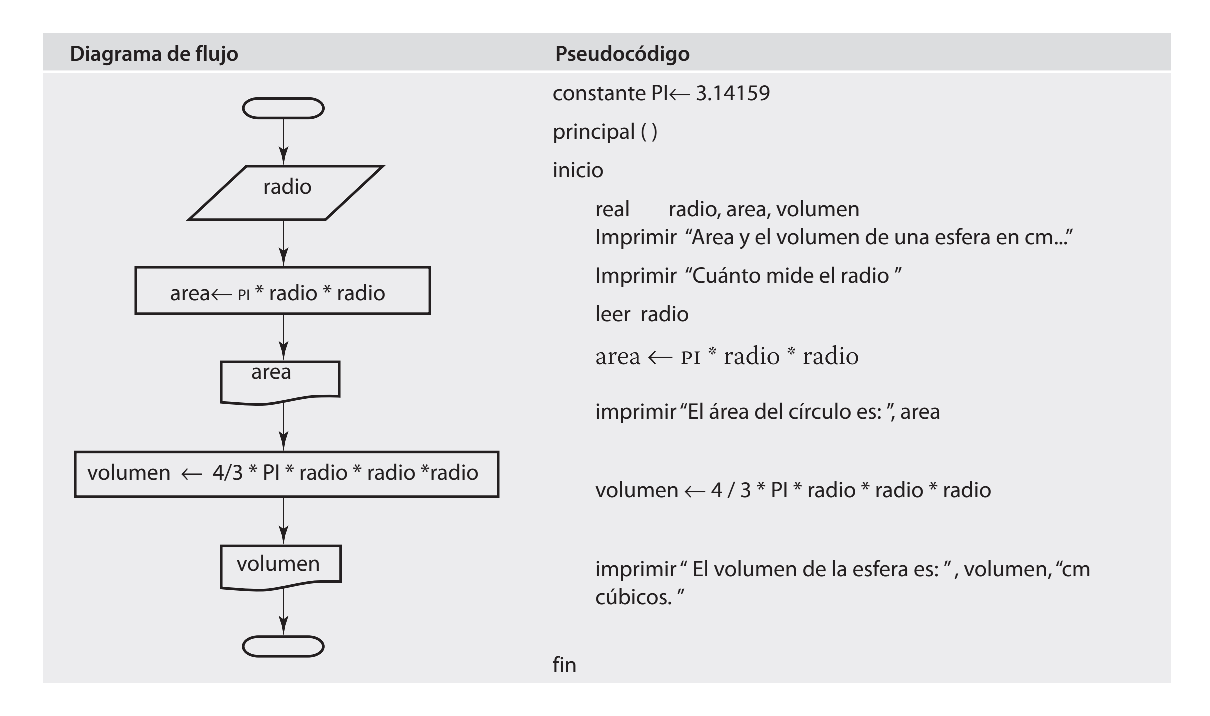
Recuerde que el tipo de dato *int* en algunos compiladores maneja rangos de valores comprendidos entre −32,768 y 32,767, por lo que si deseamos calcular el área con un lado de 200 tendremos que utilizar el tipo *long*, ya que el rango de valores está entre −2,147'483,648 y 2,147'483,647 y queda el programa:

```
main()
{
   long lado;
   char unid[12];
   printf("Dame las unidades:  ");
   scanf("%s",&unid);
   printf("Escribe la longitud de un lado del cuadrado:  ");
   scanf("%li",&lado);
   printf("El área del cuadrado es igual a: %li %s",lado*lado, unid);
   getch();
   return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:
El formato para un entero largo tanto en scanf como en printf es li.
Dame las unidades: metros
Escribe la longitud de un lado del cuadrado: 200
El área del cuadrado es igual a: 40 000 metros

Qué sucede si la longitud de un lado del cuadrado es: 123456789. El área del cuadrado es igual a: −1757895751. Este valor es inválido porque no alcanza en 2 bytes de memoria que ocupa un dato de tipo int (entero) por lo cual tendríamos que utilizar el tipo float (real) cuyos valores están comprendidos entre -3.4×10^{-38} y $3.4 \times 10^{+38}$.
En el siguiente ejercicio utilizaremos el tipo *float.*

Ejercicio 3. Área del círculo y el volumen de una esfera con el mismo radio.



Pseudocódigo

constante PI← 3.14159
principal ( )
inicio
    real   radio, area, volumen
    Imprimir "Area y el volumen de una esfera en cm..."
    Imprimir "Cuánto mide el radio "
    leer radio
    area ← PI * radio * radio
    imprimir "El área del círculo es: ", area
    volumen ← 4 / 3 * PI * radio * radio * radio
    imprimir " El volumen de la esfera es: " , volumen, "cm cúbicos. "
fin

Aquí añadimos un nuevo concepto, el de constante, definido anteriormente; el valor de *PI* será igual a 3.14159 durante todo el programa. Se declaran tres variables de tipo real, *radio* es la única variable de entrada que se necesita para obtener el área y el volumen; ésta es la que se pide en la instrucción *leer radio*, el valor se guarda en la variable *radio* y luego se utiliza el radio para calcular el área; cuando en la expresión aparece *PI* éste es sustituido por el valor asignado en la declaración de constantes, así que se multiplica 3.14159 por el valor del *radio* al cuadrado y éste se guarda en la variable *area*; este valor se muestra en pantalla con la instrucción *imprimir*, luego se realiza el cálculo para el *volumen* utilizando el mismo valor que se tiene almacenado en la variable *radio* y en *PI* y se guarda en la variable *volumen*, que luego se imprime en la última instrucción.

Ejemplos



Si radio = 5 imprimirá: *El área del círculo es: 78.53 y El volumen de la esfera es: 523.59.*

La instrucción *imprimir*, imprimirá todo aquello que aparezca entre comillas dobles (“), se refiere a la etiqueta que desea mostrar, más el resultado esperado, es decir la variable(s), pero cuando se ejecuta el programa sólo se mostrarán los valores de estas variables y no el nombre que les asignamos a éstas.

La variable *area* no debe acentuarse, recuerde que un identificador no permite caracteres especiales.

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
#define PI    3.14159
main()
{
   float radio,area,volumen;
   printf(“Este programa calcula el área y el volumen de una esfera (en cm)...\n”);
   printf(“¿Cuánto mide el radio?\n”);
   scanf(“%f”,&radio);
   area  = PI  * radio  * radio;
   printf(“El área del círculo es:  %.2f cms.\n”,area);
   volumen = (4./3) * PI * (radio * radio * radio);
   printf(“El volumen de la esfera es:  %.2f cm cúbicos.”,volumen);
   getch();
   return  0;
}
```

#define *PI* 3.14159

Definimos la constante *PI* con el valor de 3.14159.

float radio,area,volumen;

Es una sentencia declarativa que indica que se van a utilizar las variables llamadas *radio*, *area*, y *volumen*, las cuales son tipo *float*. Este tipo de dato se utiliza para declarar variables numéricas que pueden tener decimales.

En la fórmula para el volumen es importante que el numerador o el denominador tengan el punto decimal, ya que en caso contrario el resultado será: 84.82 (ver conversión de tipos).

printf(“El área del círculo es: %.2f cms.\n”,area);

El código %f se usa para representar variables del tipo *float*. El valor de la variable *area* se imprimirá con dos decimales *%.2f*.

Ejercicio 4. Hipotenusa de un triángulo rectángulo.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
	`#include <math.h>`
principal ()	`main()`
inicio	`{`
entero a, b	`int a,b;`
real c	`float c;`
imprimir "Cateto adyacente al ángulo:"	`printf("Cateto adyacente al ángulo: ");`
leer a	`scanf("%d",&a);`
imprimir "Cateto opuesto al ángulo: "	`printf("Cateto opuesto al ángulo:  ");`
leer b	`scanf("%d",&b);`
c ← raizcuad (a * a + b * b)	`c = sqrt(a * a  + b * b  );`
imprimir "La hipotenusa mide: ", c	`printf("La hipotenusa mide: %.2f",c);`
	`getch();`
	`return  0;`
fin	`}`

En el ejercicio anterior se declaran dos variables de entrada enteras *(a, b)* y una de salida *(c)* que es real ya que la fórmula calcula la raíz cuadrada.

Se solicitan el cateto adyacente y el opuesto y se almacenan en *a* y *b*, respectivamente. En la instrucción donde se realiza el cálculo, primero se realiza la expresión que se encuentra dentro de los paréntesis y por el orden de prioridad de los operadores se calculan los productos y luego la suma, y a este resultado se le calcula la raíz cuadrada con la función *raizcuad()* que ya está programada para muchos lenguajes y el resultado se almacena en C. Este resultado se imprime en pantalla con la última instrucción.

Las variables son de tipo *float* debido a que el resultado de la raíz será una cantidad real, ya que en caso de ser tipo entero nos imprimirá cero. Como podemos ver, se pueden escribir varias instrucciones en una sola línea, pero no es conveniente que sean demasiadas, ni tampoco salirnos del recuadro del editor de lenguaje C. Incluimos la librería *math.h* ya que aquí encontramos todas las funciones matemáticas, entre ellas *sqrt*.

Una posible salida en pantalla, después de ejecutar el programa sería:

Cateto adyacente al ángulo: 3
Cateto opuesto al ángulo: 4
La hipotenusa mide: 5.00

Se declaran las dos variables (cm es tipo real, *pulg* es tipo entero) y se solicita el valor de las pulgadas y se guarda en la variable *pulg*, se realiza el cálculo y se almacena en la variable *cm*, para luego imprimir el resultado.

Ejemplos



Si *pulg* = *8* se imprimirá en pantalla *8 pulgadas* = *20.320 centímetros*.

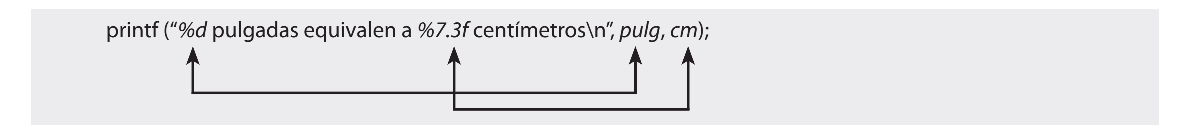
Ejercicio 5. Conversión de pulgadas a centímetros.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real cm; entero pulg	`float cm; int pulg;`
imprimir "Conv. de pulgadas a centímetros"	`printf("Conv. de pulgadas a centímetros \n");`
imprimir "Dame las pulgadas"	`printf("Dame las pulgadas ");`
leer pulg	`scanf("%d",&pulg);`
cm ← pulg * 2.54	`cm = pulg * 2.54;`
imprimir pulg ,"pulgadas = ", cm,"centímetros"	`printf ("%d pulgadas = %7.3f centímetros\n", pulg,cm);`
	`getch();`
	`return 0;`
fin	`}`

Ahora en el código en C analicemos la línea:

printf ("%d pulgadas equivalen a %7.3f centímetros ",pulg,cm);

printf() tiene tres argumentos. El primero es la cadena de control, con dos códigos de formato: *%d* y *%f*. *printf()* necesita dos argumentos adicionales, los cuales se acoplan en orden, de izquierda a derecha, con los códigos de formato. Se usa *%d* para la variable *pulg* y *%f* para la variable *cm*, y esta última variable se imprimirá en un ancho de siete espacios y con tres decimales.



Ejercicio 6. Una tienda vende libros a $100, cuadernos a $15.50 y plumas a $2.35. Calcule el monto total de una venta según el número de artículos vendidos.

Pseudocódigo

```
constante PLIB ← 100
constante PCUAD ← 15.5
constante PPLUM ← 2.35
principal ()
inicio
    entero libros, cuadernos, plumas
    real venta
    imprimir "¿Cuántos libros vendió?"
    leer libros
    imprimir "¿Cuántos cuadernos vendió?"
    leer cuadernos
    imprimir "¿Cuántas plumas vendió?"
    leer plumas
    venta ← libros * PLIB + cuadernos * PCUAD + plumas * PPLUM
    imprimir "La venta de libros fue de:", libros * PLIB
    imprimir "La venta de cuadernos fue de:", cuadernos * PCUAD
    imprimir "La venta de plumas fue: ", plumas * PPLUM
    imprimir "La venta total del día fue de:", venta
fin
```

En este ejercicio se utilizan tres constantes (PLIB, PCUAD, PPLUM) para guardar el precio de los libros, cuadernos y plumas, respectivamente, de tal manera que si el precio de éstos cambiara sólo lo tendremos que modificar en la declaración de las constantes y de esta manera el algoritmo no varía pero el resultado adquiere el nuevo valor.

Se declaran tres variables de entrada enteras (libros, cuadernos y plumas) que almacenarán la cantidad de cada producto que desea comprar. Son enteras, ya que no es posible comprar uno y medio libros, por ejemplo.

En la siguiente instrucción se realiza el cálculo que se almacena en la variable *venta*, que es real ya que algunos precios son reales y al hacer el producto y la suma el resultado será real. En el momento de hacer la instrucción (*libros* * PLIB + *cuadernos* * PCUAD + *plumas* * PPLUM) los valores de las constantes son sustituidos por los declarados al inicio y la instrucción se calcula (*libros * 100 + cuadernos * 15.5 + plumas * 2.35*) solicitando primero al usuario la cantidad de *libros*, *cuadernos* y *plumas*.

Ejemplos



Si son libros = 2, cuadernos = 4 y plumas = 3 se sustituyen los valores en la fórmula y se almacena el resultado en venta = 2 * 100 + 4 * 15.5 + 3 * 2.35, si se realizan primero los productos quedan venta = 200 + 62 + 7.05, y luego la suma donde el resultado venta = 269.05 y se imprimirá: *La venta total del día fue de: 269.05*.

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
#define PLIB  100
#define PCUAD  15.5
#define PPLUM  2.35
main()
{
    int libros,cuadernos,plumas;
    float venta;
    printf("¿Cuántos libros vendió?\n");
    scanf("%d",&libros);
    printf("¿Cuántos cuadernos vendió?\n?");
    scanf("%d",&cuadernos);
    printf("¿Cuántas plumas vendió?\n");
    scanf("%d",&plumas);
    venta  = libros  * PLIB  + cuadernos * PCUAD  + plumas  * PPLUM;
    printf("La venta de libros fue de: $%d\n",libros  * PLIB);
    printf("La venta de cuadernos fue de:$%.2f\n",cuadernos  * PCUAD);
    printf("La venta de plumas fue:$  %.2f\n",plumas  * PPLUM);
    printf("La venta total del día fue de:$  %.2f\n",venta);
    getch(); return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

¿Cuántos libros vendió?

1

¿Cuántos cuadernos?

2

¿Cuántas plumas?

3

La venta de libros fue de: $100

La venta de cuadernos fue de: $31.00

La venta de plumas fue: $7.05

La venta total del día fue de: $138.05

En el ejercicio 7 se declaran dos variables de entrada, *base* y *alt*, y tres variables de salida, *area, peri* y *diag*. Se solicitan los datos de entrada con la instrucción *imprimir "Dame la base y la altura"*, almacenando estos dos valores en *base* y *alt*. Posteriormente se realizan los cálculos para el área, perímetro y diagonal, almacenándose en *area*, *peri*, *diag*. Es importante recalcar que las dos variables de entrada, *base* y *alt* se utilizan en las tres fórmulas, ya que son las medidas para el mismo rectángulo. Después se imprimen los tres resultados.





Si la *base* = 4 y la *alt* = 3, se utilizan los valores en la fórmula y se almacena en *area* = 4 * 3, en *peri* = 2 * 4 + 2 * 3, y por último *diag* = raizcuad (4 * 4 + 3* 3), se imprime: *El área es: 12, El perímetro es: 14, La diagonal es: 5*.

Ejercicio 7. Área, perímetro y diagonal de un rectángulo.

Pseudocódigo	Lenguaje C
	`#include <math.h>`
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real base, alt, area, peri, diag	`float base, alt, area, peri, diag;`
imprimir "Dame la base y la altura "	`printf ("Dame la base y la altura ");`
leer base, alt	`scanf ("%f%f", &base, &alt);`
area ← base * alt	`area = base * alt;`
peri ← 2* base + 2* alt	`peri = 2* base + 2* alt;`
diag ← raizcuad(base*base + alt*alt)	`diag = sqrt(base*base+alt*alt);`
imprimir "El área es:",area	`printf ("El área es: %5.2f",area);`
imprimir "El perímetro es:", peri	`printf ("El perímetro es: %5.2f",peri);`
imprimir "La diagonal es:",diag	`printf ( "La diagonal es: %5.2f",diag);`
	`getch(); return 0;`
fin	`}`

En lenguaje C existe la función predefinida *sqrt* para calcular la raíz cuadrada de un valor (vea el capítulo de programación modular), dicha función la encontramos en la librería math.h; por ello tuvimos que anexar dicha librería al principio del código en C.

Ejercicios complementarios de la estructura de control secuencial en pseudocódigo y diagrama de flujo

Ejercicio 1. Complete el siguiente algoritmo que calcula el área del trapecio e identifique las variables de entrada y salida.

```
principal ()
inicio
    real ________________ , basemenor, altura, area
    imprimir "Escribe la longitud de la Base mayor en cm: "
    ________________ basemayor
    imprimir "Escribe la longitud de la Base menor en cm: "
    leer ________________
    ________________ "Escribe la altura"
```

```
    leer altura
    _________________← (basemayor + basemenor) * altura / 2
    imprimir "El área del trapecio es: cm.", area
fin
```

Identifique las variables de:

Entrada ____

Salida ____

Ejercicio 2. Complete el siguiente algoritmo que calcula el área y el perímetro de un triángulo isósceles e identifique las variables de entrada y salida.

```
principal ()
inicio
    _________________ base, altura, lado, area, perimetro
    imprimir "Cuánto mide la base del triángulo isósceles "
    leer _________________
    imprimir "Cuánto mide la altura del triángulo"
    leer altura
    area ← _________________
    _________________ ← raizcuad ((base/2)*(base/2)+altura*altura)
    perimetro ← _________________
    _________________ "El área del triángulo es: ",area
    imprimir "y el perímetro es: ", _________________
fin
```

Identifica las variables de:

Entrada ____

Salida ____

Ejercicio 3. ¿Qué realiza el siguiente pseudocódigo?

```
principal ()
inicio
    real x, y, z
    imprimir "Escriba dos números"
    leer x, y
    z← x * x + 2 * x * y + y * y
    imprimir "El Resultado es," z
fin
```

Explicación:

Identifique las variables de:

Entrada ____

Salida ____

Ejercicio 4. ¿Qué realiza el siguiente pseudocódigo?

```
principal ()
inicio
    real x, y, z, r
    imprimir "Escriba tres números"
    leer x, y, z
    r ← ( x + y + z ) / 3
    imprimir "El resultado es", r
fin
```

Explicación:

Identifique las variables de:

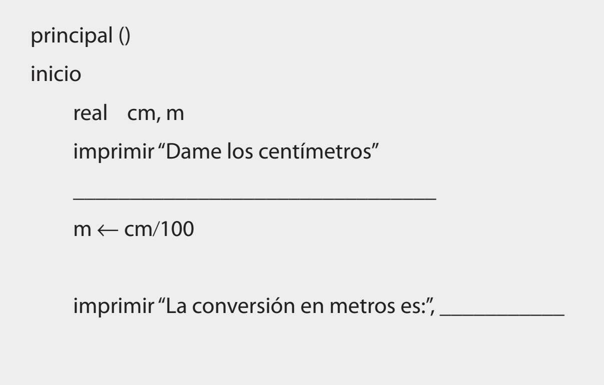
Entrada ______________________________________

Salida ____

Ejercicio 5. Utilizando los algoritmos 3 y 4 realice las siguientes pruebas de escritorio. ¿Qué valor final almacena la variable z y la variable r?

Para el ejercicio 3	Para el ejercicio 4
x = 5 y = 3 z =_____________	x = 8 y = 12 z = 7 r = _____________

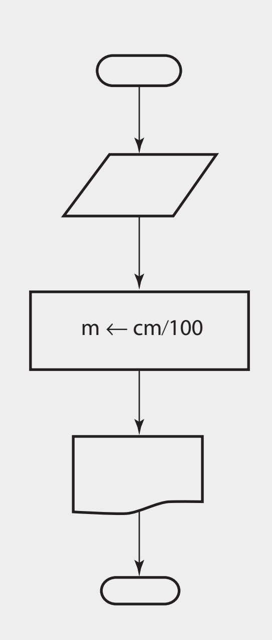
Ejercicio 6. Complete el siguiente pseudocódigo y diagrama de flujo para convertir centímetros a metros.



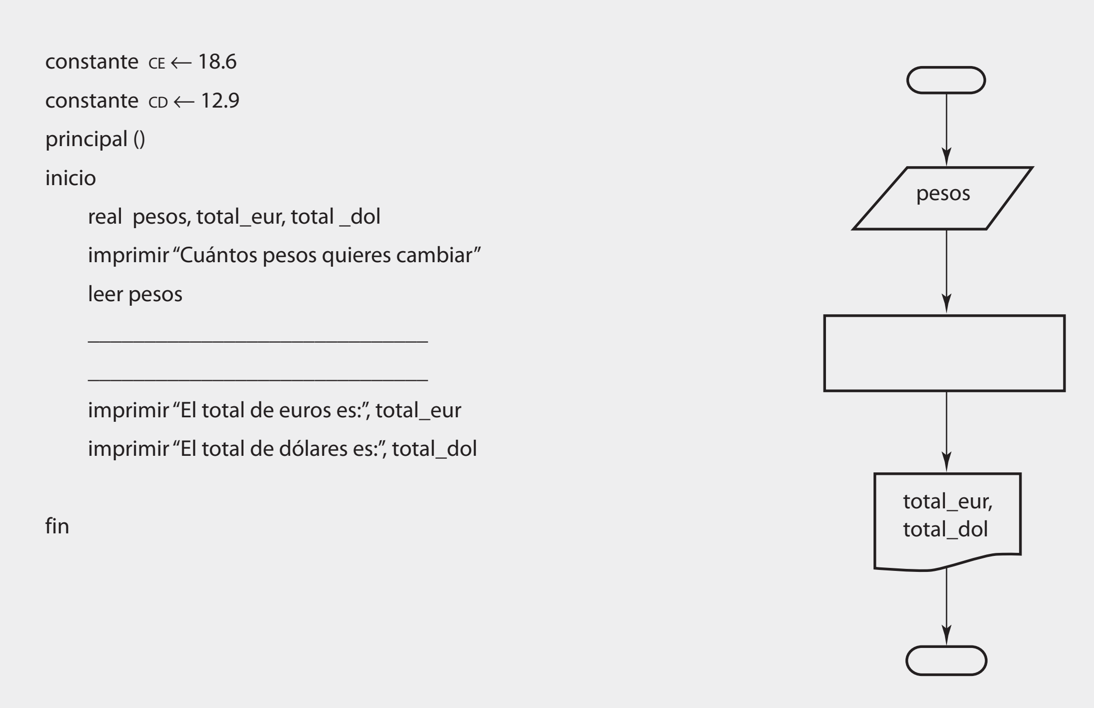
```
principal ()
inicio
    real  cm, m
    imprimir "Dame los centímetros"
    _________________________________
    m ← cm/100

    imprimir "La conversión en metros es:", ___________


fin
```



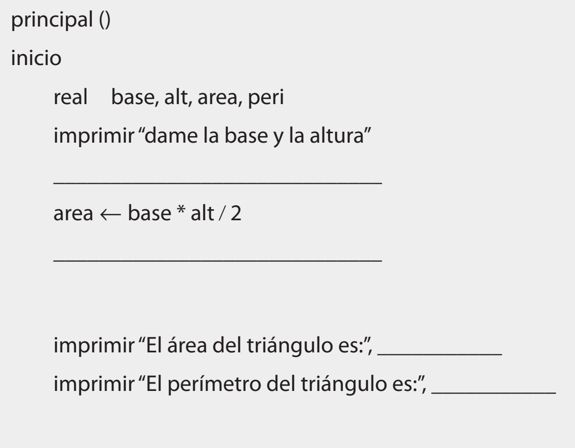
Ejercicio 7. Complete el siguiente pseudocódigo y diagrama de flujo para convertir pesos a dólares y euros, utilizando dos constantes, euro = 18.6 y dólares = 12.9.



```
constante  CE ← 18.6
constante  CD ← 12.9
principal ()
inicio
      real  pesos, total_eur, total _dol
      imprimir "Cuántos pesos quieres cambiar"
      leer pesos
      ______________________________
      ______________________________
      imprimir "El total de euros es:", total_eur
      imprimir "El total de dólares es:", total_dol

fin
```

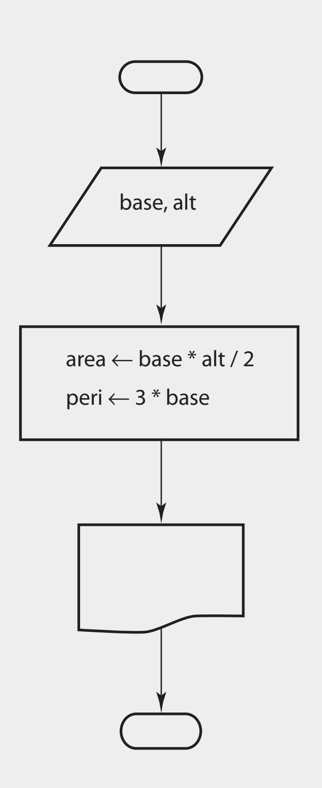

Ejercicio 8. Complete el siguiente pseudocódigo y diagrama de flujo para encontrar el área y perímetro de un triángulo equilátero.



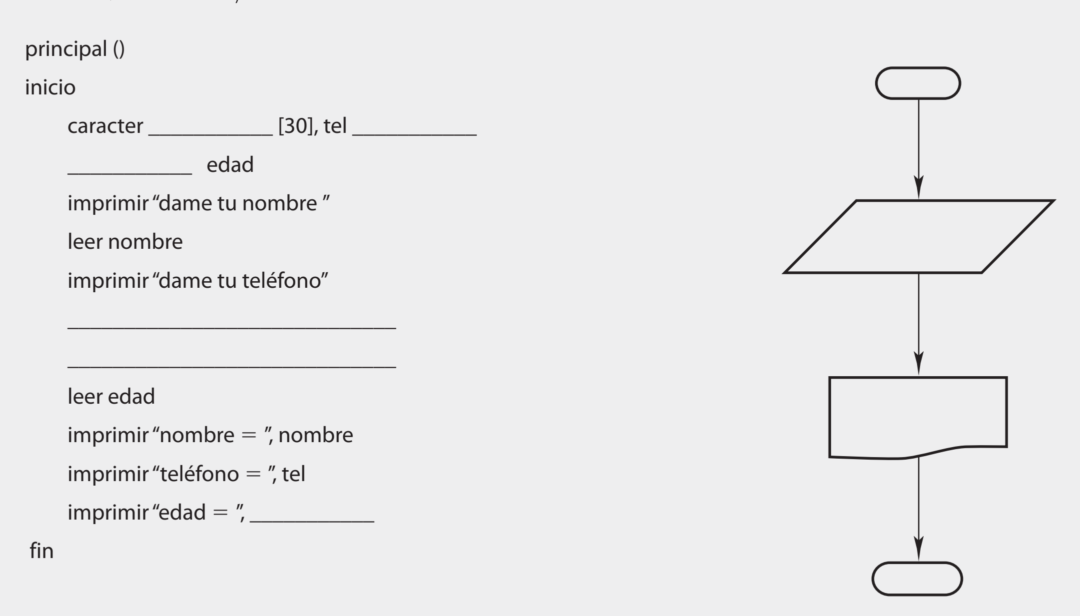
```
principal ()
inicio
      real    base, alt, area, peri
      imprimir "dame la base y la altura"
      ______________________________
      area ← base * alt / 2
      ______________________________

      imprimir "El área del triángulo es:", ___________
      imprimir "El perímetro del triángulo es:", ___________

fin
```



Ejercicio 9. Complete el siguiente pseudocódigo y diagrama de flujo para imprimir sus datos personales: nombre, domicilio y edad.



```
principal ()
inicio
    caracter ___________ [30], tel ___________
    ___________ edad
    imprimir "dame tu nombre "
    leer nombre
    imprimir "dame tu teléfono"
    ______________________________
    ______________________________
    leer edad
    imprimir "nombre = ", nombre
    imprimir "teléfono = ", tel
    imprimir "edad = ", ___________
fin
```



Ejercicios complementarios de la estructura de control secuencial en lenguaje C

Parte I. Completar los espacios faltantes en los siguientes programas

Ejercicio 1. Calcule la suma, la resta, la multiplicación y la división de dos números reales.

```
#include <conio.h>
#include <stdio.h>
main()
{
   __________
   printf("Dame el primer número ");
   scanf("___________",&n1);
   printf("Dame el segundo número ");
   scanf("%f",___________);
   printf ("La suma = %f\n",n1 + n2);
```

```
   printf ("La resta = %f\n",___________);
   printf ("La multiplicación = %f\n",n1 * n2);
   printf ("La división = %f\n",n1 / n2);
   ___________
   ___________
}
```

Si ya completó las líneas faltantes, cuál será la salida si:

Dame el primer número 4.

Dame el segundo número 2.

Ejercicio 2. Leer una cantidad de pesos (número entero) y calcular a cuántos billetes de 1000, 500, 200, 100, 50, 20 y monedas de 10, 5, 1 peso equivale.

```
#include <stdio.h>
#include <conio.h>
main()
{
   int Pesos;
   printf ("Introduce los pesos:   ");  scanf("%d",&___________);
   printf ("%d  billete(s) de 1000\n", Pesos / 1000);
   Pesos = Pesos % 1000;
   printf ("%d  billete(s) de 500\n", Pesos / 500);
   Pesos = ________________
   printf ("%d  billete(s) de 200\n", Pesos / 200);
   Pesos = Pesos % 200;
   printf ("%d  billete(s) de 100\n", Pesos / 100);
   Pesos = Pesos % 100;
   printf ("%d  billete(s) de 50\n", Pesos / 50);
   Pesos = Pesos % 50;
   printf ("%d  billete(s) de 20\n", ___________);
   Pesos = Pesos % 20;
   printf ("%d  moneda(s) de 10\n", Pesos / 10);
   Pesos = Pesos % 10;
   printf ("________________", Pesos / 5);
   Pesos = Pesos % 5;
   printf ("%d  moneda(s) de 1", Pesos / 1);
   Pesos = Pesos % 1; getch();return 0;
}
```

Si ya completó las líneas faltantes, cuál será la salida si:

Introduce los pesos: 1245.

Ejercicio 3. Realice tres conversiones de unidades de longitud del sistema inglés al decimal.

```
#include <conio.h>
#include <stdio.h>
main()
{
   float in,c,y,m,ft,___________;
   printf("Dame las pulgadas = ");________________
   c = 2.54 * in;
   printf("%f pulgadas = %f centímetros\n",in,c);
   printf("Dame las yardas = ");  scanf("%f",&y);
   m = 0.9144 * y;
   printf("___________ yardas = %f metros\n",y,___________);
   printf("Dame los pies = "); scanf("%f",&ft);
   cm = 30.48 * ft;
   printf("%f pies = ___________ centímetros",___________,cm);
   getch(); return 0;
}
```

Parte II. Elaborar la codificación en Lenguaje C de los siguientes programas (realizando el cálculo e impresión respectiva):

1. La ecuación del movimiento uniformemente acelerado para mostrar el efecto del desplazamiento inicial es la siguiente: $s = s_o + v_o t + ½\, at^2$

   Donde s = desplazamiento en el tiempo (m) s_o = desplazamiento inicial (m)
   v_o = velocidad inicial (m/s) a = aceleración (m/s^2)
   t = tiempo (s)

2. Leer un sistema de ecuaciones lineales, e imprimir su solución sabiendo que:

$$ax + by = c \qquad dx + ey = f \qquad x = \frac{ce - bf}{ae - bd} \qquad y = \frac{af - cd}{ae - bd}$$

3. El volumen de la Tierra considerando que es un esferoide: $V = 4/3 = \pi a^2 b$

   Donde: a = Radio ecuatorial = 6378.137 km, b = Radio polar = 6356.752 km

4. Áreas y volúmenes de las siguientes figuras:

   Donde: Área = A, Volumen = V, Radio = r, Altura = h, Lado = l, Diámetro = D

Cubo	$A = 6l^2$	$V = l^3$
Paralelepípedo recto rectangular	$A = 2l_1l_2 + 2l_1l_3 + 2l_2l_3$	$V = l_1l_2l_3$
Cilindro recto circular	$A = 2\pi r^2 + 2\pi rh$	$V = \pi r^2 h$
Cono recto circular	$A = \pi r^2 + \pi rl$	$V = ⅓\, \pi r^2 h$
Tronco de cono		$V = ⅓\, \pi h(R^2 + r^2 + Rr)$
Barril		$V = \frac{1}{12}\, \pi h(2D^2 + d^2)$
Cilindro hueco		$V = ¼\, \pi h(D^2 - d^2)$

5. Conversión de grados a radianes y viceversa.
6. Conversión de temperatura expresada en grados Celsius a Fahrenheit: f = (9 /5)c + 32 y viceversa.
7. Tres funciones matemáticas (vea el capítulo de programación modular).
8. Tres conversiones de tiempo.




Convertir horas a días.

9. Número de segundos que hay en días (d), horas (h), minutos (m) y segundos (s).
10. Leer un número real e imprimirlo varias veces con distintas precisiones.

Ejemplos

```
4.0127:
                                   4
                                 4.0
                                4.01
                               4.013
                              4.0127
```

11. Multiplicación de dos números reales con tres cifras decimales. Imprimir el resultado en el siguiente formato:

```
0023.72*
0145.14 =
-------------
3443.08
```

12. Calificación final de un alumno en el curso de Introducción a la programación. Dicha evaluación tiene los siguientes porcentajes:

    40%, 2 exámenes departamentales
    30%, 2 exámenes parciales
    20%, tareas, trabajos y participación en clase
    10%, proyecto

13. El área de un triángulo dadas las coordenadas de tres puntos (x1,y1), (x2,y2) y (x3,y3).
14. Leer el peso en gramos (máximo 5000) y determinar el menor número de pesas que hay que poner en una balanza (con pesos: 1 g, 2 g, 5 g, 10 g, 50 g, 100 g, 200 g, 500 g, 1000 g) para equilibrar un determinado peso en gramos, introducida por teclado.
15. Leer el lado de un triángulo equilátero. Calcular el perímetro, la altura, y el área de dicho triángulo.

3.5 Estructura de control selectiva o alternativa

Es una estructura con una sola entrada y una sola salida en la cual se realiza una acción (una o varias instrucciones) de entre varias, según una condición; o se realiza una acción según el cumplimiento o no de una determinada condición. La condición puede ser simple o compuesta.

Los programas, para un mejor funcionamiento y para poder realizar un número mayor de tareas, deben permitir emplear acciones alternativas a fin de poder elegir una de ellas cuando la situación lo requiera. Por lo tanto, la ejecución de una línea o grupos de líneas del programa depende de si cumplen o *no* una condición.

Toma de decisiones

La instrucción *si (if)* nos permite tomar decisiones, podemos hacer una pregunta y la contestación sólo puede ser verdadera o falsa, es decir, sí o no.

Ejemplos



Si llueve, llevar el paraguas.

La expresión condicional

La *condición* es una expresión booleana. Si el resultado de la expresión:

- es *cero*, se considera una condición *falsa*.
- *no es cero*, se considera una condición *cierta*.

Ejemplos



x = 9; if (x) //La condición es verdadera.

if (5>7) //La condición es falsa, comparamos dos valores.

El último ejemplo es el más utilizado durante este curso.

Existen tres tipos de estructuras de control selectivas; éstas se basan en una condición o en una opción:

a) Simple *if*. *b*) Doble *if-else*. *c*) Múltiple *switch-break*.

Bloque de sentencias o instrucción compuesta

Se denomina bloque de sentencias a un conjunto de instrucciones delimitadas, por ejemplo en lenguaje C, C++, Java se utilizan llaves que abren y cierran { }.

En lenguajes como Pascal, Modula, Delphi, Ada, se utiliza Begin y End.

Estas instrucciones se toman como una sola sentencia en las estructuras de control.

Ejemplos



Pseudocódigo	Lenguaje C
inicio	{
inst 1	`inst 1`
inst 2	`inst 2`
inst 3	`inst 3`
………	……… .
inst n	`inst n`
fin	}

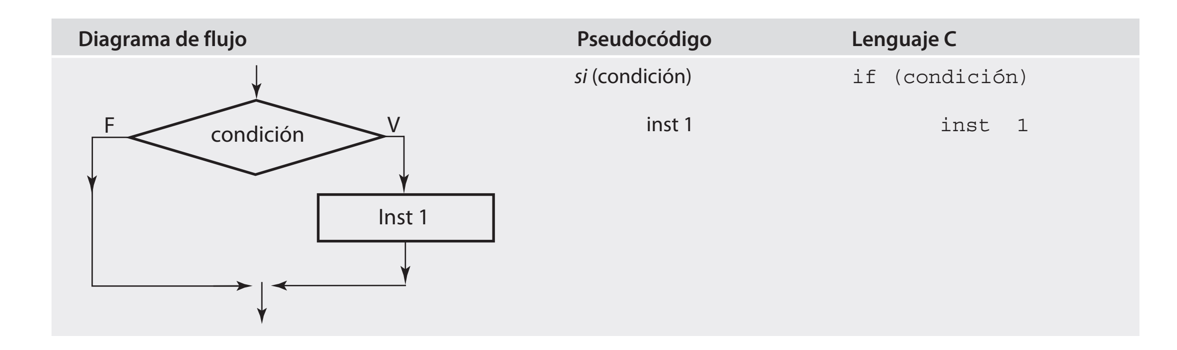
3.5.1 Estructura de control selectiva simple si (if)

Estructura de control que dirige a la computadora para ejecutar una o más instrucciones solamente si la condición es verdadera. Si la condición es falsa no realiza ninguna acción.

El término condición lo utilizaremos a lo largo de este libro para referirnos a una o más condiciones.

Existen dos formas de representarlo, dependiendo del número de instrucciones que se desean realizar si la condición se cumple:

1. Si se requiere ejecutar una sola instrucción, cuando se cumpla la condición se representa de la siguiente forma:

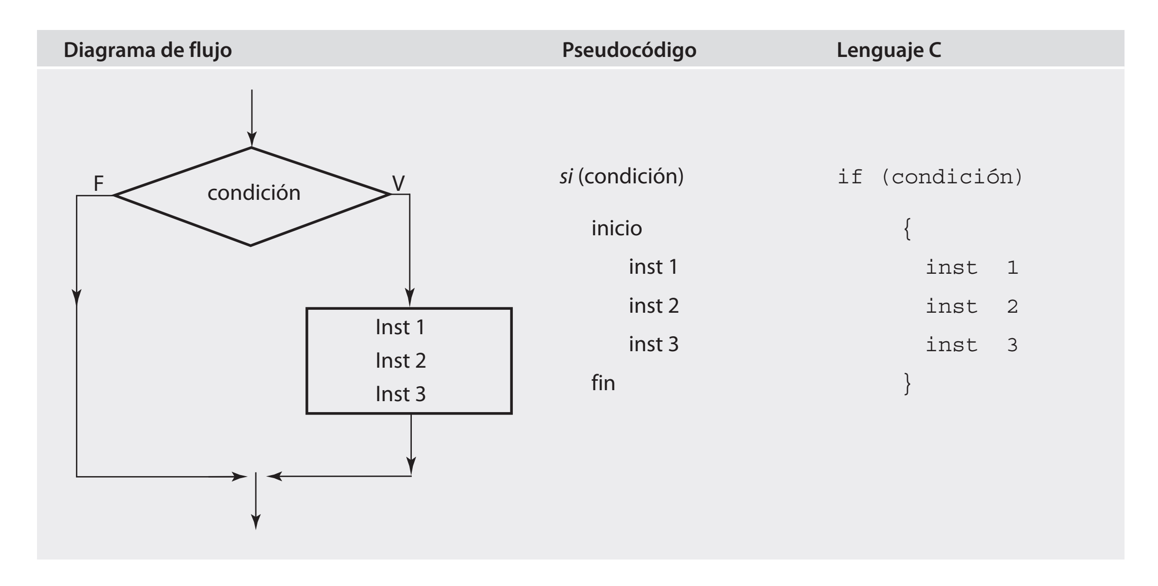
Diagrama de flujo	Pseudocódigo	Lenguaje C
	*si* (condición) inst 1	`if (condición)` `inst 1`

Si la condición se cumple se realiza la instrucción 1.

Una posibilidad en el código sería: *if* (i)

printf("%d", i/2);

2. Si se requiere ejecutar un bloque de instrucciones, cuando se cumpla la condición se representa de la siguiente forma:



Nota: Si existe más de una instrucción para realizar, es necesario utilizar inicio y fin para agrupar las instrucciones, es su alcance sintáctico, si no se usa el inicio y fin sólo se ejecuta la primera instrucción, mientras que las siguientes instrucciones se realizarán siempre. En la tabla 3.2 vemos unos ejemplos de las distintas formas que puede adoptar la "expresión" dentro de un *if*.

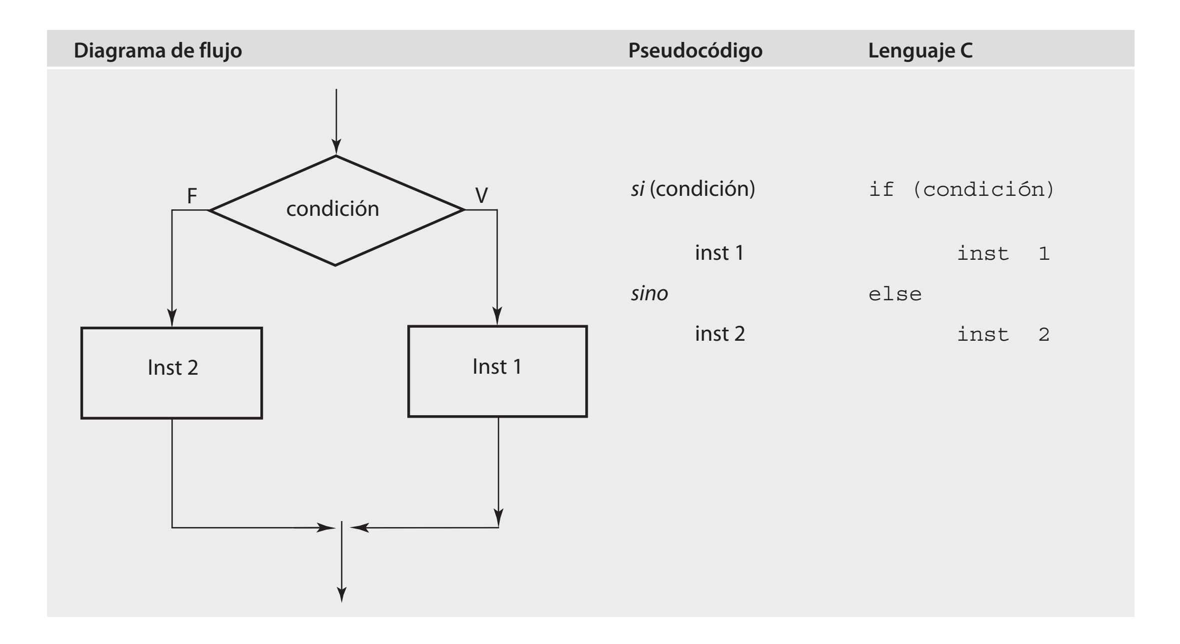
Tabla 3.2 Formas de utilizar la instrucción *if*

Ejemplos	Observaciones
if (x > y) if ((x > y) ! = 0)	Las dos expresiones son válidas y equivalentes. Es más frecuente la primera; una condición es verdadera si es diferente de cero.
if (x) if (x ! = 0)	Las dos expresiones son correctas e iguales. Es más usual la segunda expresión en este curso.
if (!x) if (x == 0)	Las dos expresiones son permitidas y semejantes. Es más común la segunda expresión; una condición es falsa si es igual a cero.
if (x = y)	La expresión es incorrecta; recuerde que el operador relacional en lenguaje C es "==", por lo que debe ser if (x == y). Un solo igual es asignación.

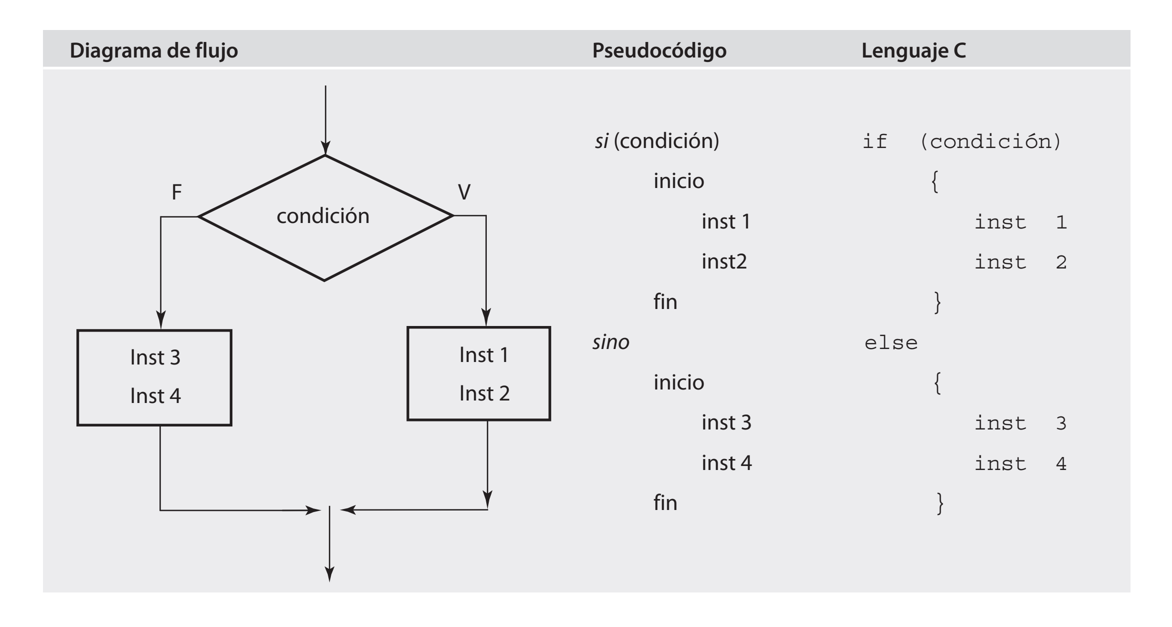
3.5.2 Estructura de control selectiva doble si/si-no (if/else)

Estructura de control que dirige a la computadora para ejecutar una acción si la condición es verdadera, y otra acción en caso de que sea falsa. Cabe mencionar que las instrucciones deberán ser diferentes en cada caso, ya que si fueran iguales no se requeriría una estructura selectiva, con la estructura secuencial se resolvería el problema.

Existen dos formas de representarlo, dependiendo del número de instrucciones que se desean realizar si la condición se cumple o no:



Si la condición se cumple se realiza la instrucción 1, pero si la condición no se cumple se realiza la instrucción 2.



Si la condición se cumple se realizan las instrucciones 1 y 2, pero si no se cumple se realizarán las instrucciones 3 y 4.

3.5.3 Anidamiento o escalonamiento si-si no-si (if-else-if)

El concepto de anidamiento es muy utilizado en la programación y consiste en insertar (anidar) una estructura dentro de otra.

La expresión "si anidados" se refiere a que podemos utilizar una sentencia *si (if)* dentro de otra sentencia *si (if)*. Esto se emplea cuando se tienen que cumplir varias condiciones para poder ejecutar una acción.

Uno de los aspectos más confusos de las sentencias *si (if)* en el lenguaje de programación es el *si (if)* anidado. Un *si (if)* anidado es una sentencia *si (if)* que es el objeto de otro *si (if)* o *si no (else)*. La razón por la que los *si (if)* anidados son tan problemáticos es que resulta confuso saber qué *else* se asocia con qué *si (if)*.

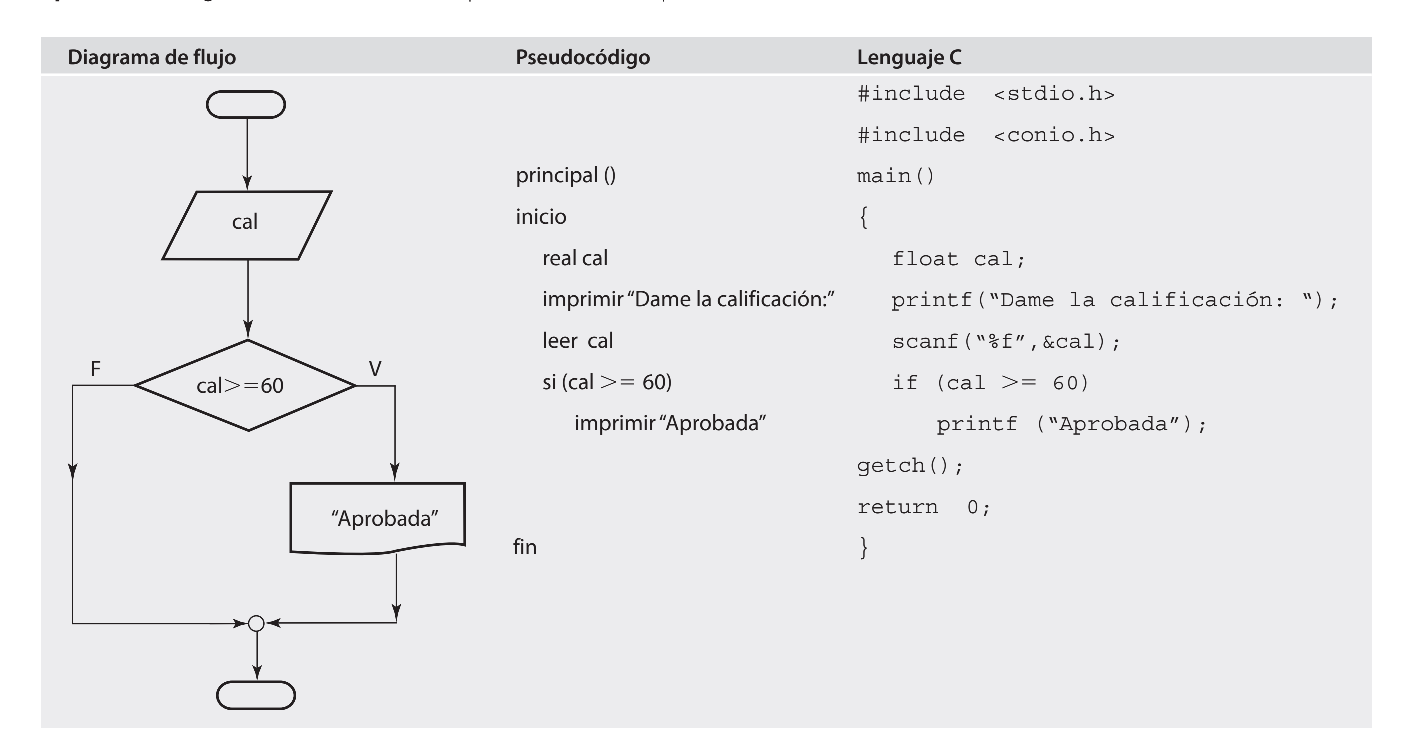
Sintaxis:

Pseudocódigo	Lenguaje C
si (condición1)	`if (condición1)`
inst1(s)	`    inst1(s);`
sino	`else`
si (condición2)	`    if (condición2)`
inst2(s)	`        inst2(s);`
sino si (condición3)	`    else if (condición3)`
inst3(s)	`        inst3(s);`
...	`        ...`
sino	`      else`
inst (s)	`    inst (s);`

La computadora evalúa la expresión condicional de arriba abajo. Tan pronto como la computadora encuentra una condición verdadera, ejecuta la instrucción(es) y salta el resto de la escalera. Si no es verdad ninguna condición, la computadora ejecutará el *else* final. Éste actúa frecuentemente como condición por defecto; es decir si fallan todas las pruebas anteriores, la computadora realiza la última instrucción(es) *else*. Si no hay *else* final y son falsas las condiciones, no se hace nada.

3.5.4 Ejercicios resueltos de la estructura de control selectiva simple si (if) y doble si/si-no (if/else)

Ejercicio 1. Según una calificación, imprimir si ésta es aprobada.

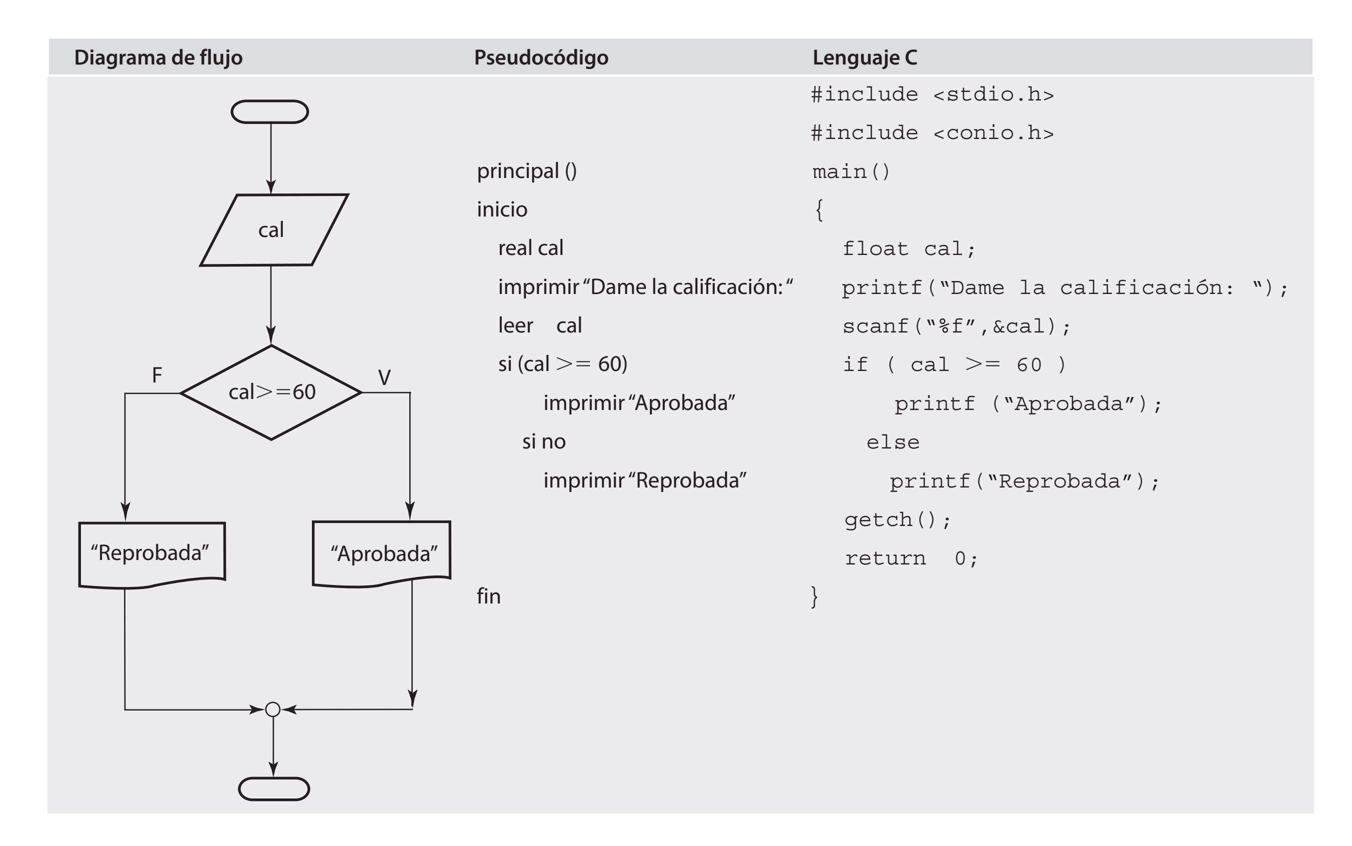


Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real cal	`float cal;`
imprimir "Dame la calificación:"	`printf("Dame la calificación: ");`
leer cal	`scanf("%f",&cal);`
si (cal >= 60)	`if (cal >= 60)`
imprimir "Aprobada"	`printf ("Aprobada");`
	`getch();`
	`return 0;`
fin	`}`

En el ejercicio 1 se utiliza la estructura selectiva simple. La *condición* de este ejemplo y la mayoría que veremos en el presente curso consisten en comparar dos valores (la variable cal y 60), y si dicha condición es

verdadera se realiza una determinada acción. Se revisa la calificación *(cal)*, si es mayor o igual que 60, se imprime *aprobada*, de lo contrario no imprime nada el programa, es decir sólo pasa por parte del programa dependiendo de una condición. Ejemplo: si $cal = 85$ el programa imprime "Aprobada", pero si $cal = 59$, el programa no imprime nada.

Ejercicio 2. Según una calificación, imprimir si es aprobada o reprobada.



Pseudocódigo

```
principal ()
inicio
   real cal
   imprimir "Dame la calificación: "
   leer   cal
   si (cal >= 60)
         imprimir "Aprobada"
      si no
         imprimir "Reprobada"
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
   float cal;
   printf("Dame la calificación: ");
   scanf("%f",&cal);
   if ( cal >= 60 )
      printf ("Aprobada");
    else
      printf("Reprobada");
   getch();
   return  0;
}
```

A diferencia del ejercicio 1, el ejercicio 2 utiliza la selectiva doble, ya que revisa la condición; si ésta se cumple (es verdadera) imprimirá "Aprobada", pero, si la condición no se cumple imprimirá "Reprobada", es decir el programa pasa por una u otra instrucción pero no por ambas. Ejemplo: si $cal = 85$ el programa imprime "Aprobada", pero si $cal = 59$, el programa imprime "Reprobada". La alternativa doble es más rápida; en este ejemplo sólo evalúa una condición, a diferencia de la simple, que evaluaría dos.

Para calcular un promedio primero se deberán sumar las n calificaciones y luego dividir el resultado entre n; en este caso específico son tres calificaciones, así que primero se le piden al usuario *cal1*, *cal2*, *cal3*. Se suman las calificaciones y el resultado se divide entre tres, guardando el resultado en la variable *promedio*, tomando en cuenta la necesidad de los *( )* en la expresión para indicarle al compilador que primero se hace la suma y luego la división. Luego se revisa la condición; si promedió > 95 se modificará el valor de *promedio* asignándole el valor de 100, pero si la calificación fuera menor o igual al 95 el valor de la variable *promedio* no se modifica, es decir queda almacenado el mismo resultado. La última instrucción muestra el resultado del promedio.

Ejemplos

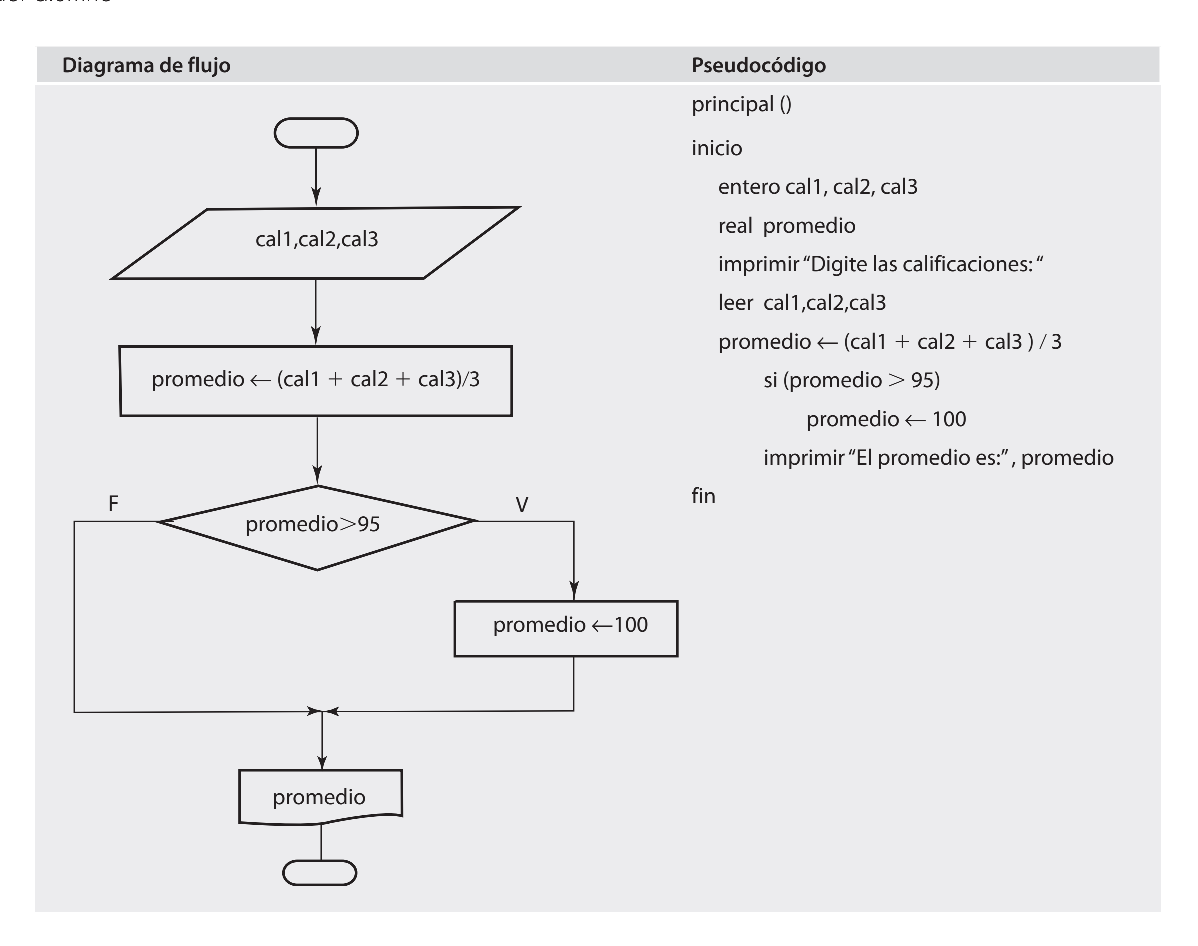


cal1 = 80, cal2 = 90, cal3 = 100. El programa imprimirá *El promedio es: 90,0* pero si *cal1 = 98, cal2 = 90, cal3 = 100* el programa imprimirá *El promedio es: 100* ya que el promedio de las tres calificaciones anteriores es de 96.0 y al ser mayor que 95 modifica el valor del promedio a 100 y es el valor que se muestra.

Nota: La última instrucción (*imprimir*) se realiza se cumpla o no la condición, ya que se encuentra fuera de la estructura de control.

Ejercicio 3. Calcule el promedio de un alumno que cursó tres materias el semestre anterior. Si su promedio es mayor que 95 se le va a asignar una calificación de 100. Imprimir cuál fue la calificación promedio del alumno

Diagrama de flujo | **Pseudocódigo**




```
principal ()
inicio
    entero cal1, cal2, cal3
    real  promedio
    imprimir "Digite las calificaciones: "
    leer  cal1,cal2,cal3
    promedio ← (cal1 + cal2 + cal3 ) / 3
        si (promedio > 95)
            promedio ← 100
        imprimir "El promedio es:", promedio
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
    int cal1,cal2,cal3; float promedio;
    printf("Digite las calificaciones del alumno:\n");
    scanf("%d  %d  %d",&cal1,&cal2,&cal3);
    promedio = ( cal1 + cal2 + cal3)/3.;//El 3. es necesario por la conversión de tipos
    if ( promedio > 95 )
         promedio = 100;
     printf("El promedio del alumno es:  %.2f",promedio);
     getch();    return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Digite las calificaciones del alumno:

95 96 95

El promedio del alumno es: 100.00

A diferencia de los programas anteriores, leemos las tres calificaciones en una sola función *scanf* (cal1 para la primera calificación, cal2 para la segunda calificación, y cal3 para la tercera calificación). Cuando el compilador llega a la línea del *scanf* debemos dar las tres calificaciones separadas con un espacio (ya que si las pegamos el compilador tomaría un solo número), o dar cada calificación y oprimir la tecla Enter. Como observamos, la sintaxis de la función *scanf* se sigue conservando. El formato *%d* para cada variable y el *&* antes de cada variable a leer.

Ejercicio 4. Imprimir si un número es par o impar.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero num	`int num;`
imprimir “Introducir un número”:	`printf(“Introducir un número:\n”);`
leer num	`scanf(“%d”,&num);`
si (num mod 2 = 0)	`if ( num % 2 == 0 )`
imprimir num “es par”	`printf(“%d es par.”,num);`
si no	`else`
imprimir num “es impar”	`printf(“%d es impar.”,num);`
	`getch();`
	`return 0;`
fin	`}`

Este ejercicio incorpora un nuevo concepto, *mod*, que sirve para ver si un número es múltiplo de otro; en este caso si la expresión *num mod 2* es igual a cero, imprimirá “el número es par”, pero si el mod es uno imprimirá “el número es impar”. No es necesario especificar las dos condiciones, ya que si no es par por consiguiente es impar. En este caso la estructura de control selectiva doble es la adecuada, ya que sólo existen dos opciones posibles (par o impar).

Ejemplos



Si *num* = *8* se imprimirá en pantalla *8 es par*, pero si *num* = *5* se imprimirá en pantalla *5 es impar*.

Ejercicio 5. Calcule el monto del pago de una persona al inscribirse en la alberca olímpica, dependiendo de si es alumno de la Universidad o no. Si es alumno se le hará un 50% de descuento tanto en la inscripción como en la mensualidad. Las cuotas sin descuento son: inscripción: 100, mensualidad: 150.

Pseudocódigo

```
constante   I ← 100
constante   M ←150
principal()
inicio
     entero tipo
     imprimir “Si la persona es alumno de la Universidad oprima (1)”
     imprimir “Si no es alumno oprima (2)”
     leer  tipo
     si  (tipo = 1)
          imprimir “El pago del alumno debe ser  de:”, I * 0.50 + M * 0.50
     si no
          imprimir “El pago si no es alumno debe ser de:”, I + M
fin
```

Otra de las aplicaciones de las estructuras selectivas son los menús, aunque en este ejercicio sólo existen dos opciones. Se puede seleccionar si es o no alumno de la Universidad; en el caso de que sea alumno se elegirá la opción 1 y este valor se guardará en la variable *tipo* o 2 si no lo es. Con base en lo que tenga la variable tipo se hacen los cálculos con o sin descuento, es decir entra a la parte verdadera *(tipo = 1)* y le haría el 50% de descuento a la inscripción y 50% a la mensualidad, pero si eligió la opción dos en el momento de revisar la condición y no cumplirse entra a la parte del si no y pagará por consiguiente el total de la inscripción y de la mensualidad.

Ejemplos



Si elige la opción 1 se imprimirá *El pago del alumno debe ser de 125,* pero si elige la opción 2 u otra se imprimirá *El pago si no es alumno debe ser de 250*.

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
#define   I   100
#define   M   150
main()
{
    int tipo;
    printf("Si la persona es alumno de la Universidad oprima  (1):\n");
    printf("Si no es alumno oprima  (2):\n");    scanf("%d",&tipo);
    if  ( tipo == 1  )
           printf("El pago del alumno debe ser de:$%.2f",I * 0.50 + M * 0.50);
    else
           printf("El pago si no es alumno debe ser de:$%d",I + M);
    getch();    return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Si la persona es alumno de la Universidad oprima (1):

Si no es alumno, oprima (2):

1

El pago del alumno es de: $125.00

Ejercicio 6. Dados los tres lados de un triángulo, imprimir si éstos pertenecen al mismo triángulo.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero a, b, c	`int a, b, c;`
imprimir "Introduzca los tres lados del triángulo:"	`printf( "Introduzca los tres lados del triángulo:");`
leer a, b ,c	`scanf("%d%d%d",&a,&b,&c);`
si (a + b > c) y (a + c > b) y (b +c > a)	`if (a+b > c && a+c >b && b+c >a)`
imprimir "Si pertenecen al mismo triángulo"	`printf ("Si pertenecen al mismo triángulo");`
si no	`else`
imprimir "No pertenecen al mismo triángulo"	`printf ("No pertenecen al mismo triángulo");`
	`getch();`
	`return 0;`
fin	`}`

Antes de querer hacer el algoritmo hay que tratar de entender el problema. Sabemos que en un triángulo siempre la suma de dos de sus lados debe ser mayor al tercer lado. Así que en una sola instrucción se revisan tres condiciones simples y en este ejercicio se incorpora el operador lógico (*y*), que nos sirve para revisar más de una condición al mismo tiempo. Se piden los dos lados del triángulo y se revisan las condiciones $(a+b>c)$ y $(a+c>b)$ y $(b+c>a)$, si las tres condiciones se cumplen se imprimirá "Sí pertenecen al mismo triángulo", pero si se cumplen dos, una o ninguna, se imprimirá "No pertenecen al mismo triángulo".

Ejemplos



$a = 3, b = 5$ y $c = 4$, sustituyendo en las condiciones $(3 + 5 > 4)$ verdadero, $(3 + 4 > 5)$ verdadero y $(5 + 4 > 3)$ verdadero; en este caso imprimirá *Sí pertenecen al mismo triángulo;* en otro caso si $a = 2$, $b = 4$ y $c = 9$ sustituyendo en las condiciones $(2 + 4 > 9)$ falso, $(2 + 9 > 4)$ verdadero y $(4 + 9 > 2)$ verdadero; en este caso imprimirá *No pertenecen al mismo triángulo*, ya que una de las condiciones resultó falsa. Cabe mencionar que en el orden de prioridad de las condiciones primero se efectúan los operadores *aritméticos* (+, −, *, /), luego los *relacionales* (>, <, =, <>, >=, <=) y por último los *lógicos (y, o, not)*.

Ejercicio 7. Imprimir si un estudiante es admitido en la universidad o no con base en los promedios que tenga de la preparatoria y del examen de admisión. El promedio mínimo es de 85.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real cal_p, cal_e, promedio	`float cal_p, cal_e, promedio;`
imprimir "Promedio de la Preparatoria?:"	`printf ("Promedio de la Preparatoria?:");`
leer cal_p	`scanf("%f",&cal_p);`
imprimir "Calificación del Examen de admisión?:"	`printf ("Calificación del Examen de admisión?:");`
leer cal_e	`scanf("%f",&cal_e);`
promedio← (cal_p + cal_e)/2	`promedio = (cal_p + cal_e)/2;`
imprimir "El promedio es:", promedio	`printf ("El promedio es: %f",promedio);`
imprimir "Por lo tanto:"	`printf ("Por lo tanto:");`
si (promedio>=85)	`if (promedio >=85)`
imprimir" El estudiante es admitido."	`printf ("El estudiante es admitido.");`
si no	`else`
imprimir "El estudiante no es admitido"	`printf ("El estudiante no es admitido");`
	`getch();`
	`return 0;`
fin	`}`

En el ejercicio 7 se trabaja con la estructura secuencial (ocho primeras instrucciones) y las siguientes corresponden a la estructura selectiva. En este caso se solicitan las dos calificaciones *cal_p, cal_e* se suman y se dividen entre dos, calculando así el promedio y luego se revisa la condición si *promedio>=85* entonces serán admitidos en la universidad.

Ejemplos



si *cal_e* = *90* y *cal_p* = *93* se imprimirá "El estudiante es admitido".

Ejercicio 8. Oprimir una tecla e imprimir qué tipo de tecla es (letra mayúscula, letra minúscula, dígito o carácter especial).

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal()	`main()`
inicio	`{`
caracter dato	`char dato;`
imprimir "oprime una tecla"	`printf ("oprime una tecla");`
leer dato	`scanf ("%c", &dato);`
si (dato>=48 y dato<=57)	`if (dato >= 48 && dato <= 57)`
imprimir " Es un dígito "	`printf ("Es un dígito");`
si no	`else`
si (dato>=65 y dato<=90)	`if (dato >= 65 && dato <= 90)`
imprimir "es un letra mayúscula"	`printf ("Es una letra mayúscula");`
si no	`else`
si (dato>=97 y dato<=122)	`if(dato >= 97 && dato <= 122)`
imprimir "es una letra minúscula"	`printf("Es una letra minúscula");`
si no	`else`
imprimir" es un carácter especial"	`printf("Es un carácter especial");`
	`getch();`
	`return 0;`
fin	`}`

En el ejercicio 8 se utilizan tres estructuras selectivas dobles anidadas para representar las cuatro opciones posibles. El código ASCII de los dígitos está entre 48 y 57 (el cero '0'= 48, '1' = 49, '9' = 57), las letras mayúsculas están entre el 65 y el 90 (A = 65, B = 66, Z = 90), las letras minúsculas entre el 97 y el 122 (a = 97, b = 98 y z = 122) y los caracteres especiales están entre varios rangos: del 0 al 47, 58 al 64, 91 al 96, el último rango del 123 al 255. La primera condición revisa si la variable *dato* es un dígito; si es, imprime *Es un dígito;* si no es entonces revisa si es una letra mayúscula; si es imprime *Es una letra mayúscula*; si no es mayúscula, entonces revisa si es minúscula; si es, imprime *Es una letra minúscula*, y por consiguiente si no fue dígito, mayúscula o minúscula entonces imprime *Es un carácter especial*.

Ejemplos



Si *dato* = *ñ*, el programa revisa la primera condición y si no se cumple revisa la segunda condición, y si no se cumple revisa la tercera y si tampoco se cumple, entonces imprime *Es un carácter especial*, ya que es la última posibilidad.

Nota 1: El ejercicio anterior no se pudo resolver con la estructura selectiva múltiple ya que los posibles datos son rangos de valores y la selectiva múltiple únicamente acepta un solo valor.

Nota 2: El concepto de anidadas se refiera a que una estructura se encuentra dentro de otra del mismo tipo; considerando la primera estructura selectiva doble el *si (dato>= 48 y dato <= 57)*, la segunda *si (dato >= 65 y dato <= 90)* y la tercera *si (dato >= 97 y dato <= 122)*. El anidamiento o la anidación se utiliza en varios casos, más adelante veremos los ciclos anidados, los cuales son la base de los arreglos bidimensionales o matrices.

Ejercicio 9. Resuelva la ecuación cuadrática.

Pseudocódigo

```
principal ()
inicio
    real a, b, c, x1, x2, d
    imprimir "Dame los valores de a, b, y c"
    leer a, b, c
    si (a< >0)
    inicio
        d ← b * b − 4 * a * c
        si (d > 0)
            inicio
                x1← ( − b + raizcuad (d)) / (2 * a)
                x2 ← ( − b  − raizcuad (d)) / (2 * a)
                imprimir "Las raíces son:", x1, x2
            fin
        si no
      si (d = 0 )
      inicio
                x1← −b / ( 2 * a)
                imprimir"Raíces repetidas =",x1
       fin
       si no
              inicio
              x1 ← − b / ( 2 * a )
              x2 ←raizcuad(fabs(d))/(2*a)
              imprimir "Raíces complejas", x1, "+−", x2
             fin
        fin
        si no     imprimir "No es ecuación cuadrática"

 Fin
```

Lenguaje C

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
   float a,b,c,x1,x2,d;
   printf ("Dame los valores de a, b y c : ");
   scanf("%f%f%f",&a,&b,&c);
   if ( a != 0 )
   {
      d = b * b − 4 * a * c ;
      if ( d > 0 )
         {
           x1 = (− b + sqrt(d)) / ( 2 * a );
           x2 = (− b − sqrt(d)) / ( 2 * a );
           printf ("Las Raíces son : %f, %f ", x1,x2);
         }
      else
         if ( d == 0 )
         {
           x1 = −b/(2 * a);
           printf ("Raíces repetidas = %f ", x1);
         }
         else
          {
           x1 = −b/(2 * a);
           x2 = sqrt(fabs( d ))/ ( 2 * a );
           printf("Raíces complejas : %f +− %f", x1,x2);
         }
    }
    else printf ("No es ecuación cuadrática ");
    getch();
    return 0;
}
```

$a x^2 + b x + c$	(a, b, c)	$\frac{-b \pm \sqrt{b^2 - 4}}{2a}$
$x^2 + 5x + 2 = 0$	(1, 5, 2)	$(-5\pm\sqrt{17})/2$
$x^2 + 2x + 1 = 0$	(1, 2, 1)	−1, −1
$x^2 - 2x + 2 = 0$	(1, −2, 2)	1± i

d = b*b −4*a*c
a <> 0
V
d > 0 Raíces reales diferentes
V
F
d = 0 Raíces reales iguales
V
F
Raíces complejas
F
No es ecuación cuadrática

El algoritmo será el siguiente:

1. Declaramos las variables *a*, *b*, *c*, *x1*, *x2*, *d* como reales.
2. Introducimos los valores de *a*, *b* y *c*.
3. *Si a es diferente de cero* significa que la ecuación es cuadrática y que la podemos resolver como tal. Como tendrá más de una instrucción, necesitamos inicio ({) y fin (}):
   a) Calculamos la discriminante $d = b * b - 4 * a * c$.
   b) Si $d > 0$ entonces las *raíces son reales diferentes;* calculamos $\frac{-b \pm\sqrt{b^2-4ac}}{2a}$ no podemos calcular ± simultáneamente por lo tanto *x1* calculará con + y *x2* con−, posteriormente imprimimos el resultado de ambas variables. Como esta condición tendrá más de una instrucción, utilizamos inicio y fin.
   c) Si no se cumple la condición anterior, $d = 0$ entonces las *raíces son reales iguales o repetidas* y bastará calcular $x1=-b/(2*a)$; posteriormente imprimimos el resultado de x1. Como esta condición tendrá más de una instrucción, utilizamos inicio y fin.
   d) Si no se cumple que $d > 0$ ni tampoco que $d = 0$, entonces $d < 0$ y las *raíces son complejas*. La parte real será $x1=-b/(2*a)$ y la imaginaria será *x2*= sqrt(abs(*d*)) /(2**a*), imprimimos la parte real (*x1*) ± la parte imaginaria (*x2*) multiplicada por i. Recuerde que las raíces complejas se dan en pares conjugados. Como este último camino tendrá más de una instrucción, utilizamos inicio y fin. Si a *no es diferente de cero* (si es igual a cero) *no es ecuación cuadrática*.

En este programa encontramos un *if* ($d > 0$) dentro de otro *if* (a != 0) cuando una estructura selectiva está dentro de otra le llamamos *if anidado*.

Si resolvemos los últimos tres ejemplos tendremos:

Ejemplos



Ejem.	a	b	c	a > 0	d=b*b−4a*c	d	x1	x2
1	1	5	2	V	5*5−4*1*2 =25−8=17	>0	(−5+√17)/2=−2.5+2.06155 =−*0.4384471*	(−5−√17)/2 =−2.5− 2.06155 =−*4.561553*
2	1	2	1	V	(2*2) −4*1*1=0	=0	x1=−b/(2*a) =−2/(2*1) =−1	x1=−b/(2*a) =−2/(2*1) =−*1*
3	1	−2	2	V	(−2*−2) −4*1*2=−4	<0	−b/(2*a)=−2/(2*1)=1	sqrt(abs(−4))/(2*1)=1i

En el ejercicio 10 se declaran tres variables *h*, *m*, *s*, todas de tipo entero; primero se solicita el número de segundos, luego se revisa con la condición *si (s<0);* si la condición es verdadera imprime "error de datos" y termina el programa ya que no existen horas, minutos y segundos; si es falsa entonces revisa otra condición *si(s >3600)* ya que una hora tiene 3,600 segundos; si es verdadera realiza una división entre el total de segundos y 3,600 *h=s /3600* el resultado lo almacena en h y lo imprime, luego realiza una resta del total de segundos que capturó menos 3,600 por las horas h que fueron impresas *s=s−3600*h*, almacenándose en s solamente los segundos que quedaron fuera de las horas; posteriormente se realiza lo mismo para identificar a cuántos minutos equivalen según los restantes con la condición *si (s>60)*, ya que un minuto tiene 60 segundos; si es verdadera se realiza una división entre el total de segundos y *60 m=s/60*, el resultado se almacena en m y lo imprime, luego realiza la resta del total de segundos menos 60 por los minutos *m* que fueron impresos *s=s−m*60*, almacenándose en s solamente los segundos que quedaron fuera de los minutos. Por último se revisa la condición *si (s>0)*, imprime los segundos que no quedaron incluidos en las horas ni en los minutos. Por ejemplo, si *s*= 8,267, *h*= 8,267/3,600 *h*= 2. *s*=8,267−7,200 *s*=1067, luego *m*=1067 / 60 *m*= 17, *s*= 1067−1020 *s*= 47, *se imprime* 2, 17, 47.

Ejercicio 10. Introducir los segundos e imprimir a cuántas horas, minutos y segundos equivale.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal()	`main()`
inicio	`{`
entero h,m,s	`int h,m,s;`
imprimir "dame el número de segundos"	`printf("dame el número de segundos");`
leer s	`scanf("%i",&s);`
imprimir "Son: "	`printf("Son: ");`
si (s<0) imprimir "error de datos"	`if(s<0) printf("error de datos\a");`
si no	`else`
inicio	`{`
si(s >3600)	`if(s>3600)`
inicio	`{`
h←s /3600; imprimir h,""	`h=s/3600; printf("%i, ",h);`
s←s−3600*h	`s=s−3600*h;`
fin	`}`
si (s>60)	`if(s>60)`
inicio	`{`
m←s/60; imprimir m,""	`m=s/60; printf("%i, ",m);`
s←s−m*60	`s=s−m*60;`
fin	`}`
si (s>0)	`if (s>0)`
imprimir s	`printf("%i",s);`
fin	`} getch(); return 0;`
fin	`}`

Ejercicio 11. Realice la prueba de escritorio o corrida a mano del siguiente programa.

Pseudocódigo	Lenguaje C
	`#include<conio.h>`
	`#include<stdio.h>`
principal ()	`main()`
inicio	`{`
entero a←3,b←5,c←7	`int a=3,b=5,c=7;`
si (a+b mod 3*2>5−c/b mod a)	`if (a+b%3*2>5−c/b%a)`
inicio	`{`
a←2+3*c; b←a−c mod 4+10	`a=2+3*c; b=a−c%4+10;`
c←b+c; imprimir a,b,c	`c=b+c; printf("%d%d%d\n",a,b,c);`
fin	`}`
si no	`else`
inicio	`{`
a←7−c mod 3; b←a*b mod 4	`a=7−c%3; b=a*b%4;`
imprimir a,b,c	`printf("%d−%d−%d\n",a,b,c);`
fin	`}`
si (a<>b y b<>c)	`if (a!=b&&b!=c)`
imprimir a*b*c	`printf("%d\n",a*b*c);`
si no	`else`
imprimir a+b+c	`printf("%d\n",a+b+c);`
	`getch(); return 0;`
fin	`}`

En la tabla 3.3 vemos los resultados de la prueba de escritorio del programa anterior.

Tabla 3.2 Corrida a mano o prueba de escritorio

a	b	c	a+b mod 3*2>5−c/b mod a	a<>b y b<>c	a*b*c	a+b+c
3	5	7	3+5 mod 3*2>5−7/5 mod 3 = 3+2*2>5−1 mod 3 = 7>5−1 = V			
2+3*7=23	23−7 mod 4+10= 23−3+10=30	30+7=37	–	23<>30 y 30<>37 V y V V	23*30*37 = 25530	
a=2+3*7=23, *b*=23−7 mod 4+10=23−3+10=30 *c*=30+7=37						

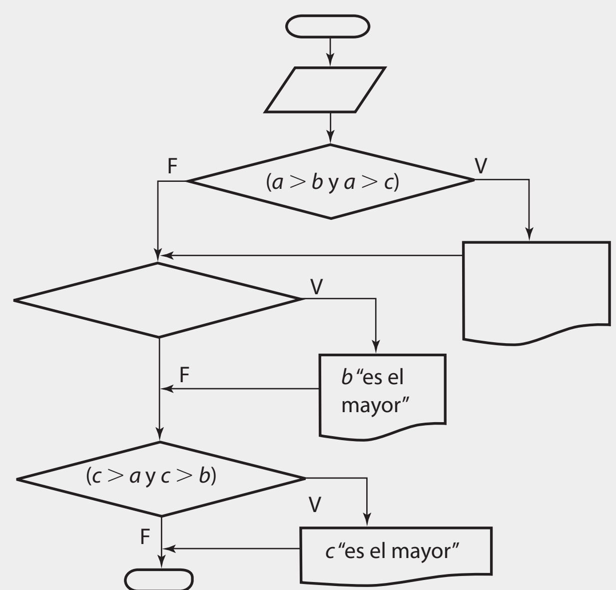
El programa contiene dos *si (if) dobles;* en el primero la condición *a+b mod 3*2>5−c/b mod a* es *verdadera*, por lo tanto calcula el primer bloque marcado con inicio y fin e imprime los valores *a=23, b=30 y c=37.* Cuando termina con el primer *si (if)*, continúa con el segundo, donde la condición *a<>b* y *b<>c* es también verdadera, imprimiendo *a*b*c=25530.*

Ejercicios complementarios de la estructura de control selectiva simple y doble en pseudocódigo y diagrama de flujo

Ejercicio 1. Identifique qué realiza el siguiente pseudocódigo y explíquelo a continuación. Además complete el diagrama de flujo respectivo.

```
principal ()
inicio
   entero     a, b, c
   imprimir "Dame 3 números"
   leer     a, b, c
   si (a > b y a > c)   imprimir   a, "es el mayor"
   si (b > a y b > c)   imprimir   b, "es el mayor
   si (c > a y c > b)   imprimir   c, "es el mayor"
fin
```

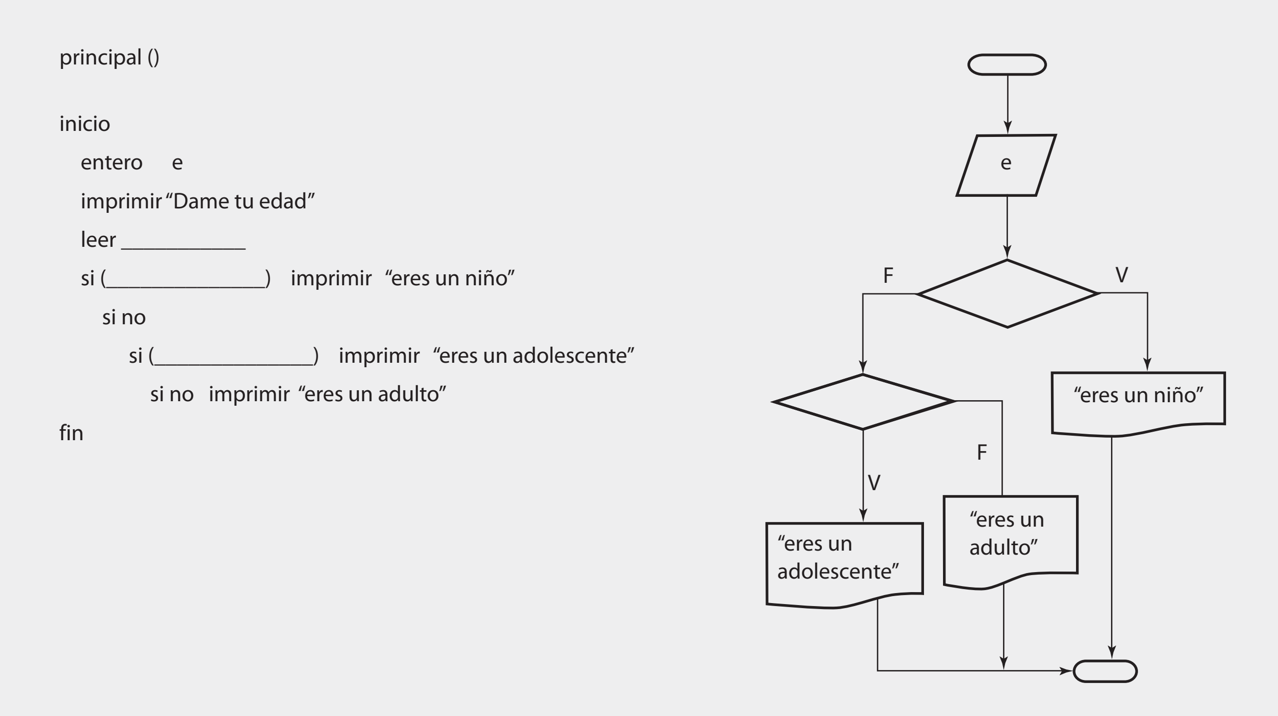
Respuesta:



Ejercicio 2. Complete los espacios vacíos para el siguiente problema. Según la edad de una persona imprimir si es niño (0-12), adolescente (13-17) o adulto (mayor o igual a 18).

```
principal ()

inicio
   entero   e
   imprimir "Dame tu edad"
   leer ___________
   si (______________)   imprimir  "eres un niño"
      si no
         si (______________)   imprimir  "eres un adolescente"
            si no  imprimir  "eres un adulto"
fin
```



Ejercicio 3. Complete los espacios vacíos para el siguiente problema. Según los tres lados de un triángulo, imprimir si es equilátero (tres lados iguales), isósceles (dos lados iguales) o escaleno (lados diferentes).

```
principal ()
inicio
     real   l1,l2, l3
     imprimir "¿Cuánto miden los tres lados?: "
     leer   l1, l2, l3
     si (______________  y  ______________)  imprimir "El triángulo es Equilátero, lados iguales."
     si no
         si (______________  o  ______________  o  ______________)
               imprimir "El triángulo es Isósceles, 2 lados iguales."
         si no  imprimir "El triángulo es Escaleno, ningún lado igual."
fin
```

Ejercicio 4. Complete el siguiente pseudocódigo que calcula la paga de un trabajador según sean las horas trabajadas y el precio que se le pagan las horas con las siguientes condiciones:

a) Si trabaja entre 30 y 40 horas se le paga la tarifa por hora normal.

b) Si trabaja más de 40 horas cada hora extra se le pagará 20% más la tarifa por hora normal.

c) Si trabaja menos de 30 horas se le descuenta 12% de su paga.

```
principal ()
inicio
    entero  horas, horas_ext
    ____________________ precio_hora, paga
    imprimir "Introduce el número de horas trabajadas"
    leer ___________________
    imprimir "_________________________________________"
    leer   precio_hora
    si (horas > 40)
       inicio
          horas_ext ← ____________________________________
          paga ← _________________________________________
       fin
    si no
       si ___________________
          paga ← horas * precio_hora
       si no
          ___________________ ← horas *   precio_hora * 0.88
    imprimir paga
fin
```

Ejercicio 5. Complete el siguiente pseudocódigo que utiliza la estructura selectiva doble anidada, para imprimir si la persona es niño, adolescente, adulto o anciano, según el rango de edades siguiente:

a) 0 a 12 es niño.

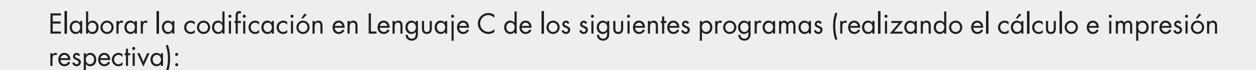
b) 13 a 17 es adolescente.

c) 18 a 80 es adulto.

d) Mayor que 80 es anciano.

```
principal ()
inicio
    entero   edad
    imprimir  "¿Cuál es la edad?"
    leer ____________________
    si (edad <= _____________ )
        imprimir "Es niño"
    si no
        __________ ( edad <= 17 )
              imprimir "es adolescente"
        si no
              si ____________________________________
                    imprimir "es adulto"
              si no
              ______________________________________
fin
```

Ejercicios complementarios de la estructura de control selectiva simple y doble en lenguaje C



Elaborar la codificación en Lenguaje C de los siguientes programas (realizando el cálculo e impresión respectiva):

1. Leer una fecha (día, mes, año) y diga si es correcta o no.
2. Leer un año e imprimir si es bisiesto o no.
3. Un millonario tenía tres hijos: Juan, Luis y Rosa. Al morir dejó el siguiente legado: A Luis le dejó 4/3 de lo que le dejó a Juan. A Juan le dejó 1/3 de su fortuna. A Rosa le dejó la mitad de lo que le dejó a Luis. Además el licenciado les cobrará por sus servicios un porcentaje de lo que reciban cada uno de herencia: si el monto es menor a la tercera parte de la herencia cobrará 3%, en caso contrario 5%. Imprimir cuánto recibirán cada heredero y el abogado.
4. El domingo de Pascua es el primer domingo después de la primera luna llena posterior al equinoccio de primavera y se determina mediante el siguiente cálculo:

   a = año % 19 b = año % 4 c = año % 7 d = (19 * a + 24) % 30

   e = (2 *b + 4 * c + 6 * d + 5) % 7 n = (22 + d + e)

   Donde n indica el número del día del mes de marzo (si n es igual o menor que 31), en caso contrario el mes de abril se obtiene como ($d+e-9$). Imprimir la fecha del domingo de Pascua a partir de un año dado.
5. Leer un número entero (entre 0 y 32 700) e imprimir el número de cifras de dicho número.
6. Leer la hora de un día (0 a 23 horas y minutos, por ejemplo para las 18:30 deberá leer 18 y 30), e imprimir la misma hora pero en formato de 12 horas mostrando las horas, los minutos y el indicador de meridiano. No es posible especificar PM cuando la hora es 0. Cuando se especifica AM también representa las horas antes del mediodía. Si se especifica PM representa las horas después del mediodía. Si se especifica AM, representa la medianoche. Si se especifica PM, representa el mediodía. 12:01 es 1 minuto después del mediodía, como 12:01 PM, mientras que 12:01 AM es 1 minuto después de medianoche. Especificar 12:01 AM es lo mismo que 00:01 o 00:01 AM. Para las 18:30 deberá imprimir 6:30 PM.
7. Leer un número entero e imprimir si es: *a*) Cero, positivo o negativo; *b*) Par o impar (cuando sea positivo); *c*) Múltiplo de 7 o no múltiplo de 7 (cuando sea impar). Para el número 35, imprimirá: Es un número positivo, es impar y es múltiplo de 7.
8. Leer 5 números e imprimir el menor y el mayor de ellos.
9. Conversión de temperatura expresada en grados Celsius a Fahrenheit : f = (9/5)c+32 y viceversa (Fahrenheit a Celsius). Permita elegir entre estas dos opciones.
10. Dados dos de los ángulos de un triángulo, calcular el tercer ángulo y determinar si el triángulo es rectángulo (un ángulo igual a 90 grados), acutángulo (sus tres ángulos menores a 90 grados) u obtusángulo (un ángulo mayor a 90 grados).
11. Leer las horas trabajadas y la paga por hora para calcular el sueldo semanal de un empleado. Cuando las horas de trabajo exceden de 40 se consideran horas extra; las primeras 8 se pagan al doble; si las horas extra exceden de 8 se pagan las primeras 8 al doble de lo que se pagan las horas normales y el resto al triple.
12. En un estacionamiento se cobra de la siguiente manera: los primeros 10 minutos son gratis, los siguientes 15 minutos subsecuentes cuestan $3, la primera hora $10 y cada hora subsecuente $5. A partir de la primera hora se cobran horas completas. Si es domingo se hace un descuento de 10% sobre el monto total. Leer los minutos y horas e imprimir cuánto se pagará.
13. Ordenar dos números con un solo if.
14. Leer 4 números. Imprimir el mayor, el menor y la suma de ambos.
15. Leer 3 números y ordenarlos de manera ascendente.

16. Existen cuatro cuadrantes dependiendo del signo de x y y. Lee las coordenadas x y y de un punto e imprime si está en los ejes de coordenadas, en el origen o bien en qué cuadrante se encuentra localizado.

	Y
Segundo cuadrante	Primer cuadrante
(0, 0)	Origen
Tercer cuadrante	Cuarto cuadrante

17. Leer el número de días transcurridos del año e imprimir a qué mes pertenece. Por ejemplo si se captura 32 se imprimirá el mes es febrero.
18. Leer la fecha de nacimiento (día, mes y año) de una persona y la fecha actual (día, mes y año), para posteriormente imprimir la edad al día de hoy. Por ejemplo usted tiene 19 años 7 meses y 14 días.
19. Resolver el ejercicio 8 resuelto con una versión diferente, por ejemplo los valores de la condición pueden ir entre apóstrofos.
20. Leer el nombre de un determinado mes e imprimir cuántos días tiene dicho mes. Utilizar la función strcmp (vea el capítulo de programación modular).
21. Calcular alguna de las cuatro operaciones básicas, con el nombre respectivo. Por ejemplo suma. Utilizar la función strcmp (vea el capítulo de programación modular).

3.5.5 Estructura de control selectiva múltiple segun_sea (switch)

Esta estructura selecciona entre varias posibilidades, dependiendo del valor de la expresión. Cuando en la estructura *si (if)* todas las condiciones utilizan la *igualdad* de una variable determinada con constantes predefinidas, se puede utilizar la instrucción *segun_sea (switch)*, en dicha instrucción existen más de dos opciones. La estructura *segun_sea (switch)* evalúa una expresión que puede tomar *n* valores distintos; según con cuál de estos valores coincida, se ejecutarán ciertas acciones, es decir, el programa o algoritmo seguirá un determinado camino entre los *n* posibles. Dicha expresión sólo acepta valores *enteros* o *caracteres* para entrar a la opción y el operador de relación es el igual.

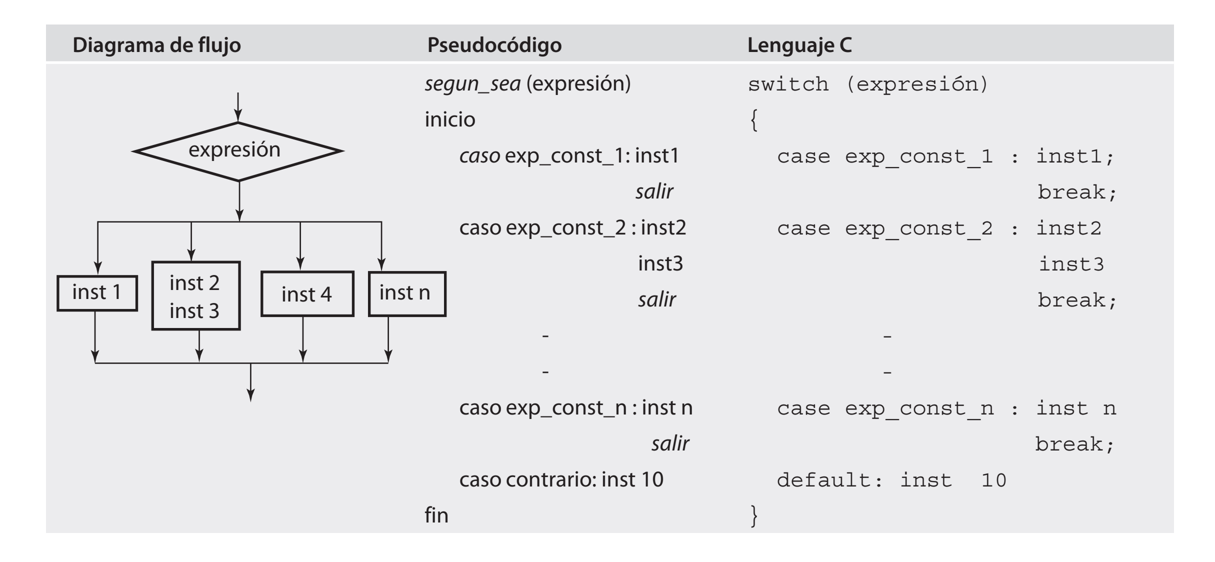


Diagrama de flujo	Pseudocódigo	Lenguaje C
	segun_sea (expresión)	`switch (expresión)`
	inicio	`{`
	caso exp_const_1: inst1	`case exp_const_1 : inst1;`
	salir	`break;`
	caso exp_const_2 : inst2	`case exp_const_2 : inst2`
	inst3	`inst3`
	salir	`break;`
	-	-
	-	-
	caso exp_const_n : inst n	`case exp_const_n : inst n`
	salir	`break;`
	caso contrario: inst 10	`default: inst 10`
	fin	`}`

Se compara la "expresión" (puede ser una sola variable) con cada una de las opciones "const" y en el momento de encontrar una constante idéntica se ejecutan la(s) instrucción(es) correspondiente(s) a ese caso. Al

terminar de realizar las instrucciones del caso, se debe usar la palabra reservada *salir (break)* para que vaya al final de la estructura.

Si ninguno de los casos cumple con la expresión, se puede definir un caso por omisión, que también puede tener instrucciones; la computadora ejecutará la sentencia *caso contrario (default)*. El *default* es opcional, si no está presente no se hace nada.

Hay tres puntos que debemos considerar en la sentencia *segun_sea (switch)*:

1. Se diferencia del *si (if)* en que el primero sólo puede comprobar por igualdad, mientras que la expresión condicionada del *si (if)* puede ser de cualquier tipo.
2. No puede tener dos constantes en los casos con idénticos valores en el mismo *segun_sea (switch)*.
3. La sentencia *segun_sea (switch)* es más eficiente que el *si anidado* si-si no-si (if-else-if).

Notas:

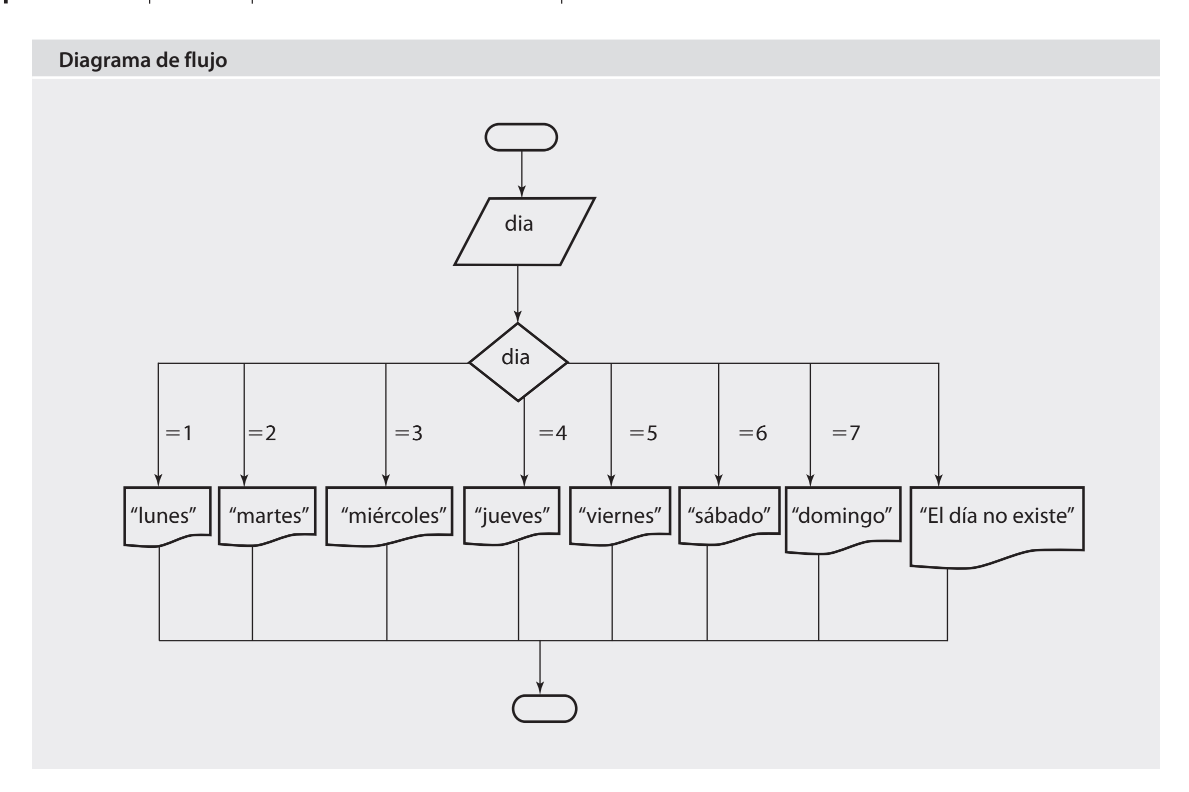
- Si cada caso tiene varias instrucciones no es necesario agruparlas con llaves, ya que el *salir (break)* termina el caso.
- Se utilizará *segun_sea (switch)* para manejar un menú. Un menú nos muestra en pantalla todas las opciones que podemos realizar con nuestro algoritmo o programa.

3.5.6 Estructura de control selectiva segun_sea (switch) anidada

Se puede tener un *segun_sea (switch)* que es parte de una secuencia de instrucciones de otro *segun_sea (switch)*. Incluso si las constantes del *caso (case)* interior contienen valores comunes, no se tendrán conflictos. Se pueden anidar sentencias *segun_sea (switch)* con el fin de implementar rutinas donde las opciones tienen subdivisiones como en el caso de los submenús.

3.5.7 Ejercicios resueltos de la estructura de control selectiva múltiple segun_sea (switch)

Ejercicio 1. Imprimir a qué día de la semana corresponde en número.



Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero dia	int dia;
imprimir "Escriba el número del día: "	printf("Escriba el número de día: \n");
leer dia	scanf("%d",&dia);
segun_sea (dia)	switch(dia)
inicio	{
caso 1: imprimir "El día ",dia," es Lunes"	case 1: printf("El día %d es Lunes",dia);
salir	break;
caso 2: imprimir "El día ",dia," es Martes"	case 2: printf("El día %d es Martes",dia);
salir	break;
caso 3: imprimir "El día ",dia," es Miércoles"	case 3: printf("El día %d es Miércoles",dia);
salir	break;
caso 4: imprimir "El día ",dia," es Jueves"	case 4: printf("El día %d es Jueves",dia);
salir	break;
caso 5: imprimir "El día ",dia," es Viernes"	case 5: printf("El día %d es Viernes",dia);
salir	break;
caso 6: imprimir "El día ",dia," es Sábado"	case 6: printf("El día %d es Sábado",dia);
salir	break;
caso 7: imprimir "El día ",dia," es Domingo"	case 7: printf("El día %d es Domingo",dia);
salir	break;
caso contrario: imprimir "El día no existe"	default: printf("El día no existe");
fin	}
	getch();
	return 0;
fin	}

Este tipo de selectiva múltiple nos sirve para elegir una entre varias opciones. En el ejercicio anterior el usuario introduce un número del 1 al 7 y se almacena en la variable *dia*, posteriormente revisa si el *día* capturado pertenece a alguno de los siete casos e imprimirá a qué día de la semana pertenece; la instrucción *salir (break)* que se escribe a un lado, le indica al programa que si ya entró en algún caso, no continúe con lo siguiente, es decir que se salga de la estructura *segun_sea (switch)*, y el programa pasará el control a la siguiente instrucción que se encuentre después del fin del *segun_sea (switch)*. Si el usuario introduce un valor menor a 1 o mayor que 7 no entra a ninguno de los casos, sino que entrará a la parte *caso contrario (default)* e imprimirá "El día no existe".

Ejemplos



Si *dia= 3* el programa imprimirá "El día 3 es Miércoles", si *dia = 9* el programa imprimirá "El día no existe". El formato para imprimir el día dentro de la expresión, depende del lenguaje.

Si omitimos la instrucción *salir (break)* recorrerá todos los casos siguientes al capturado;


Ejemplos



si la variable *dia* fue 2, pasará por los casos 3, 4, 5, 6 y 7. Observamos que cada caso no necesita inicio ni fin debido a que el *break* es el fin de cada uno de ellos.

Es importante recordar que la variable que utiliza el *segun_sea (switch)* sólo puede ser algún tipo entero o caracter *(char);* en nuestro ejemplo la variable *dia* es tipo *int*. Si hubiéramos elegido un tipo real.

Ejemplos



float dia en lugar de *int dia*, el compilador nos enviaría el siguiente mensaje: "Error Línea 10: Switch selection expression must be of integral type".

Ejemplos



Dos posibles salidas si ejecutamos el programa son:

Escriba el número de día:

2

El 2 corresponde a martes.

Escriba el número de día:

21

El día no existe.

Ejercicio 2. Proporcione el precio de un auto, seleccionando la opción de entre cinco diferentes.

Diagrama de flujo

Pseudocódigo

```
constante  A  85,000
constante  B  110,000
constante  C  152,000
constante  D  180,000
constante  E  170,000
principal ()
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
inicio
    entero opcion
    imprimir  "Elija la marca del auto para darle el precio "
    imprimir  "1: Nissan, Tsuru. 2: Golf"
    imprimir  "3: Jetta. 4: Dodge, Stratus."
    Imprimir  "5: Dodge, Neón";  leer opcion
    segun_sea  (opcion)
    inicio
        caso 1:  imprimir  "El Nissan, Tsuru cuesta :$ ",A; salir
        caso 2:  imprimir  "El VW, Golf cuesta:$ ",B; salir
        caso 3:  imprimir  "El VW, Jetta cuesta:$ ",C; salir
        caso 4:  imprimir  "El Dodge, Stratus cuesta:$",D; salir
        caso 5:  imprimir  "El Dodge, Neón cuesta:$ ",E; salir
    fin
fin
```

En el ejercicio anterior, similar al número 1, incluye el concepto de constante para que se imprima según sea el caso el precio del carro que haya seleccionado. La variable opcion almacena un número del 1 al 5 y mostrará el precio correspondiente. Por ejemplo, *si opcion = 4* el programa imprimirá *El Dodge, Stratus cuesta: $180 000;* si no se hubiera puesto el salir después del caso 4, el programa también imprimiría *Dodge, Neón cuesta: $ 170 000;* en este caso sólo imprime la primera leyenda. Otro ejemplo: *si opcion = −1* el programa imprimirá No existe la opcion −1

Lenguaje C

```c
#include  <stdio.h>
#include  <conio.h>
#define A  85000
#define B  110000
#define C  152000
#define D  180000
#define E  170000
main()
{
    int opcion;
    printf("Elija la marca del auto para darle el precio\n");
    printf("1: Nissan, Tsuru.  \n  2: VW, Golf.\n");
    printf("3: VW, Jetta.\n  4: Dodge, Stratus.\n");
    printf("5: Dodge, Neón.\n");
    scanf("%d",&opcion);
    switch(opcion)
    {
       case 1: printf("El Nissan, Tsuru cuesta:$  %li\n",A);break;
       case 2: printf("El VW, Golf cuesta:$  %li\n",B);break;
       case 3: printf("El VW, Jetta cuesta:$  %li\n",C);break;
       case 4: printf("El costo del Dodge, Stratus es:$  %li\n",D);break;
       case 5: printf("El Dodge, Neón cuesta:$  %li\n",E);break;
       default: printf("No existe la opción %i",opcion);
    }
  getch(); return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Digite una opción para darle el precio del auto que desea
1: Nissan, Tsuru.
2: VW, Golf.
3: VW, Jetta.
4: Dodge, Stratus.
5: Dodge, Neón.
−9
No existe la opción −9

Recuerde que el formato del *printf* para cada caso es *ld* o *li*, debido a que el precio de cada vehículo está contemplado dentro del entero largo. En el *default* la variable opcion no es un entero largo, por lo tanto no requiere de la *l* en su formato de impresión. No es conveniente que los costos lleven "," ya que se omitirán los últimos tres ceros.

Ejemplos



Si #define A 85,000 se imprimirá sólo $85.

Ejercicio 3. Calcule el área de un triángulo, un círculo, o de un cuadrado utilizando un menú.

Diagrama de flujo

"A) Área del triángulo"
"B) Área del círculo"
"C) Área del cuadrado"

opcion

= A, a
= B, b
= C, c

x , y
x
x

Area ← x * y /2
Area ← PI * cuad (x , 2)
Area ← x * x
"La opcion que digito no existe"

Area

Pseudocódigo

```
constante   PI   3.14159
principal ()
inicio
    caracter  opcion;    entero    x, y;    real area
    imprimir    "Este programa calcula las siguientes áreas en cm2."
    imprimir    "Elige una opción"
    imprimir    "A) Área de un triángulo"
    imprimir    "B) Área de un círculo"
    imprimir    "C) Área de un cuadrado";  leer  opcion
    segun_sea  (opcion)
    inicio
        caso 'A': caso 'a': imprimir "Digite la base y la altura:";    leer    x , y
                        area ←x * y / 2;    salir
        caso 'B': caso 'b': imprimir "¿Cuál es el radio del círculo?";    leer    x
                        area ←PI * cuad (x , 2); salir
        caso 'C': caso 'c': imprimir "¿Cuánto mide un lado del cuadrado?";    leer    x
                        area ←x * x;    salir
        caso contrario: imprimir "La opción que digito no existe."
    fin
    imprimir "El Área es: ", area, "cm2"
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define  PI  3.14159
main()
{
   char opcion; int x,y; float area;
   printf("Este programa calcula las siguientes áreas en cm2:\n");
   printf("Elija una opción:\n");
   printf("A) Área de un triángulo.\n");
   printf("B) Área de un círculo.\n");
   printf("C) Área de un cuadrado.\n");
   switch  (opcion=getchar())
   {
     case 'A': case 'a': printf("Digite la base y la altura:\n"); scanf("%d%d",&x,&y);
                         area=(x*y)/2; break;
     case 'B': case 'b': printf("¿Cuál es el radio del círculo?\n"); scanf("%d",&x);
                         area=PI*pow(x,2); break;
     case 'C': case 'c': printf("¿Cuánto mide un lado del cuadrado?\n"); scanf ("%d",&x);
                         area=x  * x; break;
     default:printf("La opción que digito no existe.\n");
   }
   printf("El Área es:  %.2f cm2",area);
   getch();    return  0;
}
```

El ejercicio anterior incluye otra característica de la estructura selectiva múltiple: la de hacer menús, ya que sirve para elegir una opción entre varias. Este ejercicio utiliza caracteres en lugar de números enteros para elegir la opción, muestra también que el carácter puede ser en mayúscula o minúscula, ya que cada uno de éstos tiene diferente códigos ASCII (A = 65, a = 97). El carácter va entre apóstrofos. Se declaran las variables *x, y* de tipo entero y se utilizarán para calcular alguna de las tres áreas, teniendo en cuenta que la estructura *segun_sea* realiza solamente una de ellas y no todas. El usuario elige la *opción* que es una variable de tipo caracter ya que almacenará una letra y dependiendo de la letra que elija se realizará el área del menú; cabe mencionar que los tres cálculos de áreas se almacenarán en la variable *area*, para que cuando se termine la estructura *segun_sea* se imprima el valor en lugar de imprimirlo dentro de cada una de las opciones. También se incorpora la función *cuad (x,2)* que eleva el valor de *x* al cuadrado, pero también sirve para elevar cualquier número a cualquier potencia.

Ejemplos 

si opcion = B el programa entrará a calcular el área del círculo, *si x = 3*, la variable *area = 3.14159*9* almacena este valor para luego imprimir *El Área es: 28.27*. Si la *opcion =a* el programa entra al caso A, y solicita la base y la altura, que las almacena en las variables *x = 4* y *y = 6;* se realiza el cálculo y la variable almacenará *area =12*, para luego imprimir *El Área es: 12*.

En lenguaje C utilizamos la función *getchar(): switch (opcion=getchar())* donde la variable *opcion* almacenará la tecla que oprimamos. Se incorpora la función *pow(x,2)* que eleva el valor de *x* al cuadrado, pero también sirve para elevar cualquier número a cualquier potencia; por lo tanto necesitamos incluir la librería math.h, ya que *pow* es una función matemática.

Ejercicio 4. Imprimir qué tipo de alumno es según su calificación, considerando que si el alumno tiene 100 se imprima Excelente. Si tiene entre:

a) 90 y 99 se imprima Muy Bueno
b) 80 y 89 se imprima Bueno
c) 70 y 79 se imprima Regular
d) 60 y 69 se imprima Malo
e) 0 y 59 se imprima Reprobado

Pseudocódigo

```
principal ()
inicio
    entero   cal
    imprimir  "¿Cuál es la calificación?";   leer cal
    si (cal <>100)
        si (cal >= 90)  cal=90
        si no
            si (cal >=80)  cal = 80
            si no
                si (cal >=70)   cal = 70
                si no
                    si (cal >= 60)  cal = 60
                    si no cal = 50
    segun_sea   (cal)
    inicio
        caso 100: imprimir "Excelente!"; salir
        caso 90:  imprimir "Muy bueno"; salir
        caso 80:  imprimir "Bueno"; salir
        caso 70:  imprimir "Regular"; salir
        caso 60:  imprimir "Malo"; salir
        caso 50:  imprimir "Reprobado"; salir
        caso  contrario:  imprimir "Calificación no válida"
    fin
fin
```

Una desventaja que tiene el *segun_sea* para el lenguaje C es que no permite utilizar rangos, así que trabajaremos con la estructura selectiva doble anidada para almacenar en la variable *cal* una nueva calificación que permita luego utilizar la estructura selectiva múltiple e imprimir qué tipo de alumno es; aunque en este ejercicio en particular se pudo haber impreso qué tipo de alumno es desde la selectiva doble según la calificación del alumno, pero en otros ejercicios puede ser útil este proceso. En este programa utilizamos varios si *(if) anidados* donde la calificación dentro de un rango se redondea a múltiplos de 10, en el siguiente ejemplo el 68 se redondea a 60 y en el caso 60 imprime *Malo*.

Ejemplos



si *cal* = *76* entrará al *cuarto si anidado*, en el cual la nueva calificación será *cal* = *70;* de esta manera al entrar a la estructura *segun_sea*, imprimirá *Regular*.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
    int cal;
    printf("¿cuál es la calificación?\n"); scanf("%d",&cal);
```

(*continúa*)

(*continuación*)

Lenguaje C

```
    if (cal!=100)
            if (cal>=90) cal=90;
               else
                   if (cal>=80) cal=80;
                       else
                           if (cal>=70) cal=70;
                              else
                                  if (cal>=60) cal=60;
                                     else cal=50;
    switch  (cal)
    {
       case 100: printf("Excelente!");break;
       case 90: printf("Muy bueno");break;
       case 80: printf("Bueno");break;
       case 70: printf("Regular");break;
       case 60: printf("Malo");break;
       case 50: printf("Reprobado");break;
       default: printf("Calificación no válida");
    } getch(); return 0;
}
```

Ejercicios complementarios de la estructura de control selectiva múltiple en pseudocódigo

Ejercicio 1. Complete el siguiente pseudocódigo que calcula la paga de un trabajador según sean las horas trabajadas y el precio por hora para el trabajo que realice, entre las siguientes opciones:

a) Intendencia: $35 *b*) Asistente: $50 *c*) Administrativo: $80
d) Directivo: $120 *e*) Gerente: $150

```
principal ()
inicio
   caracter  puesto; ________________  horas, paga_total
   imprimir "Introduce el número de horas trabajadas "; leer ________________
   imprimir "a) intendencia  b) asistente  c) administrativo"
   imprimir "d) directivo  e) gerente"
   "elige el puesto";  leer ________________
   ________________ (puesto)
   inicio
         caso 'a': paga_total  ← horas * 35; salir
         caso 'b': ________________  ← horas * 50; salir
         caso '__': paga_total  ← horas * 80; salir
         caso 'd': paga_total  ← ________________ * 120; salir
         caso 'e': paga_total  ← ____________________________________; salir
         caso  contrario: imprimir "no existe ese puesto"
```

(*continúa*)

(*continuación*)

```
            fin
    imprimir "La paga es : " , ______________________________________
fin
```

Ejercicio 2. Complete el siguiente pseudocódigo que nos dice, dependiendo el día y el mes, qué signo del zodiaco es.

```
principal ()
inicio
    entero dia, mes
    imprimir "Escribe tu día de nacimiento en dígitos del (1-31) "
    leer dia
    imprimir  "Escribe tu mes de nacimiento en dígito del (1-12) "
    leer mes
    segun_sea _________________
     inicio
         caso 1:  si (dia < 22)   imprimir "Capricornio "
                    si no imprimir "Acuario"
                    salir
         caso 2:  si (dia < 22)   imprimir "______________________________________"
                    si no imprimir "Piscis"
                    salir
         caso 3:  si (dia < 22)   imprimir "Piscis "
                    si no imprimir "______________________________________"
                    salir
         caso 4:  si (dia < 22) imprimir "Aries "
                    si no imprimir "______________________________________"
                    salir
         caso 5:  si (dia < 22) imprimir "______________________________________"
                    si no imprimir "Géminis"
                    salir
         caso 6:  si (dia < 22) imprimir "______________________________________"
                    si no imprimir  "Cáncer"
                    salir
         caso 7:  si (dia < 22) imprimir   "Cáncer "
                    si no imprimir    "______________________________________"
                    salir
         _________________ si (dia < 22) imprimir  "Leo"
                                si no imprimir  "Virgo"
                                salir
         caso 9:  si (dia < 22) imprimir  "______________________________________"
                    si no imprimir   "______________________________________"
                    salir
         caso 10:   _________________ imprimir "Libra "
                    si no imprimir  "escorpión"
                    salir
         caso 11: si (dia < 22) imprimir   "escorpión "
                    _________________   imprimir "sagitario"
                    salir
         caso 12: si (dia < 22) imprimir    "______________________________________"
                    si no imprimir   "______________________________________"
                    salir
         _________________: imprimir "No existe ese mes"
    fin
fin
```

Ejercicio 3. Explique qué realiza el siguiente pseudocódigo y qué imprime con los siguientes datos:

```
constante   PI   3.14159
principal ()
inicio
    caracter   opcion;     real    x, peri
    imprimir    "Este programa calcula los siguientes perímetros en cm."
    imprimir    "A Perímetro de un triángulo equilátero"
    imprimir     "B Perímetro de un círculo"
    imprimir    "C Perímetro de un cuadrado"
    imprimir     "Elige una opción";     leer   opcion
    segun_sea   (opcion)
    inicio
        caso 'A': caso 'a': imprimir "Digite la base:";    leer    x
                            peri ← x * 3; salir
        caso 'B': caso 'b': imprimir "¿Cuál es el radio del círculo?";    leer   x
                            peri ←  x * 2 * PI; salir
        caso 'C': caso 'c': imprimir "¿Cuánto mide un lado del cuadrado?";    leer   x
                            peri ← x * 4; salir
        caso contrario:   imprimir "La opción que digito no existe."
    fin
    imprimir "El Perímetro es: ", peri
fin
```

Respuesta:
Si opción = B, x = 5, qué imprime

Ejercicios complementarios de la estructura de control selectiva múltiple en lenguaje C

Parte I. Elaborar la codificación en lenguaje C de los siguientes programas (realizando el cálculo e impresión respectiva):

1. Leer un número entero obtenido al lanzar un dado de seis caras e imprimir el número en letras de la cara opuesta. En las caras opuestas de un dado de seis caras están los números: 1-6, 2-5 y 3-4. Si el número del dado capturado es menor que 1 o mayor que 6, se mostrará: "Un DADO no tiene ese número".
2. Capturar una letra minúscula o mayúscula e imprimir la posición respectiva por ejemplo: a=1 y la Z=27.
3. En un supermercado, se realizan descuentos por las compras a partir de unas bolitas de colores: Verde 20%, amarilla 25%, negra del 30%; la blanca no aplica ningún descuento. Leer el importe de la compra y el color de la bolita e imprimir lo que debe pagar dicho cliente.
4. El importe anual del seguro de un coche depende del modelo del coche, del color, de la edad del conductor y tiempo que tiene conduciendo. Son tres modelos de coche (A, B y C) y los precios del seguro según el color:

Modelo	*Color*	*Precio*	*Modelo*	*Color*	*Precio*	*Modelo*	*Color*	*Precio*
A	Claro	3800	B	Claro	4930	C	Claro	7570
	Oscuro	4340		Oscuro	5600		Oscuro	8250

Si el conductor tiene menos de 20 años, el precio se incrementa 25%; si tiene entre 20 y 30 años se incrementa 10%; si tiene entre 31 y 65 años el precio no se modifica; si tiene más de 65 años el precio se incrementará 15%. Además, en todos los casos si el conductor tiene menos de 2 años conduciendo con permiso, el precio se incrementará 25% adicional. Calcular el precio del seguro para un determinado modelo y un determinado conductor.

5. Leer una letra mayúscula e imprimir la letra y si ésta es recta (A,E,F,H,I,K,L,M,N,T,V,W,X,Y,Z), es curva (C,O,S,U,Q) o curva y recta (B,D,G,J,P,R).
6. Leer un carácter y dos números enteros. Si el carácter leído es un operador aritmético calcular la operación correspondiente, si es cualquier otro carácter imprimir error.
7. Elaborar un menú que calcule tres conversiones de tiempo.
8. Elaborar un menú que calcule tres conversiones del sistema inglés al decimal (unidades de longitud).
9. Elaborar un menú que calcule conversión de divisas, pesos a dólares, pesos a euros, pesos a libras, libras a pesos, dólares a pesos, etcétera.
10. Imprimir la descripción correspondiente a una cantidad de hasta cuatro dígitos dada.
11. Investigar el horóscopo chino y hacer el cálculo respectivo, conociendo el mes y el año.
12. Leer metros o centímetros y posteriormente seleccionar la opción de conversión de longitud al sistema inglés: yardas, pulgadas o pies. Al final imprimir el resultado.

Parte II. Realizar la prueba de escritorio o corrida a mano los siguientes programas:

Ejercicio 1

```
main()
{
    float A=3,B=1,C=4,D;
    B=A*B/2;
    C=B+4;
    printf("%f,%f,%f\n",A,B,C);
    if (A/B-3*C<25-C*B)
                    {
                        A=2+3*C;
                        C=A+3*C/2;
                        D=A+C;
                    }
            else  if (A>10&&B!=C)
                    {
                        A=2+4*C;
                        C=A-C/2;
                        D=A+C-2;
                    }
                  else
                    {
                        A=7-C/4;
                        B=C/4-A;
                        D=A+B;
                    }
```

Ejercicio 2

```
main()
{ int a=5,b=15,c=21,d,e;
  d=a+b;  e=a*2;
  a=3*b/2-e; b=c-a;
  printf("%d,%d,%d,%d,%d\n",a,b,c,d,e);
  switch (e-(b%4*3>2))
     {
        case 1: if (c>19){
                             b=c/3;
                             e=c%2;
                             }
                        else
                             a=e+2;
                     break;
        case 3: if (b<18)
                             a=c%2;
                        else {
                             a=e*3;
                             b=a%4;
                             c=b+5;
                             }
                     break;
        default: a=d*2;
                       c=c/3;
```

(continúa)

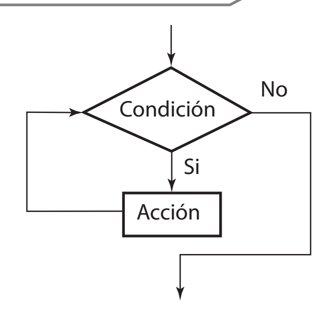
(*continuación*)

```
    printf("%f,%f,%f,%f\n",A,B,C,D);
    if (D<50) printf("%f,%f\n",A,C*2);
        else printf("%f,%f\n",B,D/2);
    getch();
    return 0;
}
```

```
        b=b%2;
}
printf("%d,%d,%d,%d,%d\n",e,d*2,b,a+4,c);
getch (); return 0;
}
```

3.6 Estructura de control repetitiva o de iteración condicional

Las computadoras están diseñadas primordialmente para aquellas aplicaciones en las cuales una operación o conjunto de ellas deben repetirse más de una vez.



La repetición de una acción (una o varias instrucciones) se lleva a cabo mientras se cumpla cierta condición; para que la acción termine, la acción misma debe modificar la(s) variable(s) de control que interviene(n) en la condición. Dicha condición puede estar predefinida como en el ciclo *desde (for)*; o *no* predeterminada, como en los bucles *mientras (while)* y *hacer-mientras (do_while)*.

Bucles, ciclo o iteración. Es un segmento de un algoritmo o programa, cuya(s) instrucción(es) se repite(n) un número conocido o indefinido de veces mientras se cumpla una determinada condición. En cada vuelta del ciclo comprueba si la condición es verdadera, rompiéndose el ciclo cuando es falsa. La condición en algún momento tiene que ser falsa ya que en caso contrario el bucle se hará infinito.

3.6.1 Contadores, acumuladores, centinelas y banderas

Contadores y acumuladores

Estas dos palabras se usan mucho en programación para referirse a variables que van incrementando o decrementando su valor a lo largo de la ejecución del algoritmo o programa. Normalmente serán de tipo numérico,

Ejemplos



Pseudocódigo	Lenguaje C
cont ← cont + 1	`cont = cont + 1;`
total ← total + y	`total = total + y;`
m ← m * 3	`m = m * 3;`

Características de la variable contador o acumulador:

1. Se debe inicializar antes de entrar al ciclo.
2. Dentro del ciclo debe aparecer a ambos lados de la asignación.

Debemos inicializarlas de forma correcta, para:

- *Sumar* o *restar* en *cero* antes de entrar al ciclo; recordar que el cero es el elemento neutro de la suma y resta.
- *Multiplicar 1*, porque si valen cero todo lo que multipliquemos por ella seguirá valiendo cero; recordar que el *1* es el elemento neutro de la multiplicación.

Contador

Un contador es una forma de controlar a un bucle. Es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción en cada repetición; dicha variable controla o determina la cantidad de veces que se repite un proceso o dato. Sintaxis:

Ejemplos



Pseudocódigo	Lenguaje C
nom_contador ←nom_contador + val_const	`nom_contador = nom_contador + val_const;`

Si en vez de *incremento* es *decremento* se coloca un *menos* en lugar del *más*.

Se debe realizar una operación de inicialización y posteriormente las sucesivas de incremento o decremento del mismo. La inicialización consiste en asignarle al contador un valor de inicio. Se situará antes y fuera del bucle.

Ejemplos



Pseudocódigo	Lenguaje C
entero contador ← 1	`int contador = 1; //variable con valor inicial de  1`
contador ← contador+1	`contador++; o ++contador; o contador+=1;`

Acumulador o totalizador

Un acumulador realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante. Es una variable que acumula sobre sí misma un conjunto de valores, para de esta manera tener la acumulación de todos ellos en una sola variable. Es una variable que almacena cantidades resultantes de operaciones sucesivas.

Sintaxis:

Ejemplos



Pseudocódigo	Lenguaje C
nom_acumula ←nom_acumula + valor_var	`nom_ acumula = nom_ acumula + valor_var;`

Pseudocódigo	Lenguaje C
entero acumulador ← 0	`int acumulador = 0;`
acumulador ← acumulador + valor	`acumulador = acumulador + valor; o` `acumulador += valor;`

Centinela

El centinela es una variable que inicia con un valor, luego dentro de un bucle este valor cambia, haciendo falsa la condición del ciclo y por lo tanto indicará el fin del ciclo (el usuario puede determinar cuándo hacerlo). La repetición controlada por centinela se considera como una repetición indefinida (se desconoce el número de repeticiones).

Ejemplos



Un usuario puede introducir S o N para indicar si desea continuar o no. El bucle terminará cuando la respuesta del usuario sea "n" o "N". También se podrían manejar números.

Bandera

Una bandera (FLAG, INTERRUPTOR O MARCA), es una variable que puede tomar sólo dos valores opuestos, generalmente: 1 (verdadero) o 0 (falso), a lo largo de la ejecución del algoritmo o programa. Es muy utilizado en búsquedas, veremos un ejemplo en un arreglo unidimensional en el siguiente capítulo.

Ejemplos



Pseudocódigo	Lenguaje C
int band ← 0	`int band = 0;`
bandera ← 1	`bandera = 1;`

Existen tres estructuras repetitivas: *mientras (while)*
hacer_mientras (do_while)
desde (for)

Ahora describiremos cada una de ellas.

3.6.2 Estructura de control repetitiva mientras (while)

Al ejecutarse la instrucción *mientras (while),* se evaluará la expresión booleana suministrada en los paréntesis (condición), y si su valor es verdadero (distinto de cero) se realizará el ciclo o bucle (una o varias instrucciones). Después, la condición es reevaluada y se procede de la misma manera. Cuando la condición se vuelve falsa (es decir, cero), en la siguiente evaluación se dará por terminado el ciclo *mientras (while).*

Si la condición nunca se vuelve cero, el ciclo nunca terminará y, dependiendo de las instrucciones incluidas en el bucle, se generaría un error de ejecución que detendría el programa, o podría ser que el programa itere indefinidamente hasta ser detenido en forma manual.

En esta estructura no se conoce necesariamente el número de veces que entrará al ciclo, ya que esto dependerá de la condición definida.

Diagrama de flujo	Pseudocódigo	Lenguaje C
condición / F / V / Inst 1, Inst 2, …., Inst n	*Con una instrucción* mientras (condición) Inst 1 *Con varias instrucciones* mientras (condición) inicio Inst 1 Inst 2 - Inst n fin	`Con una instrucción` `while (condición)` `        Inst 1` `Con varias instrucciones` `while (condición)` `{` `    Inst 1` `    Inst 2` `    -` `    Inst n` `}`

donde *condición* es cualquier expresión numérica, relacional o lógica.

Características:

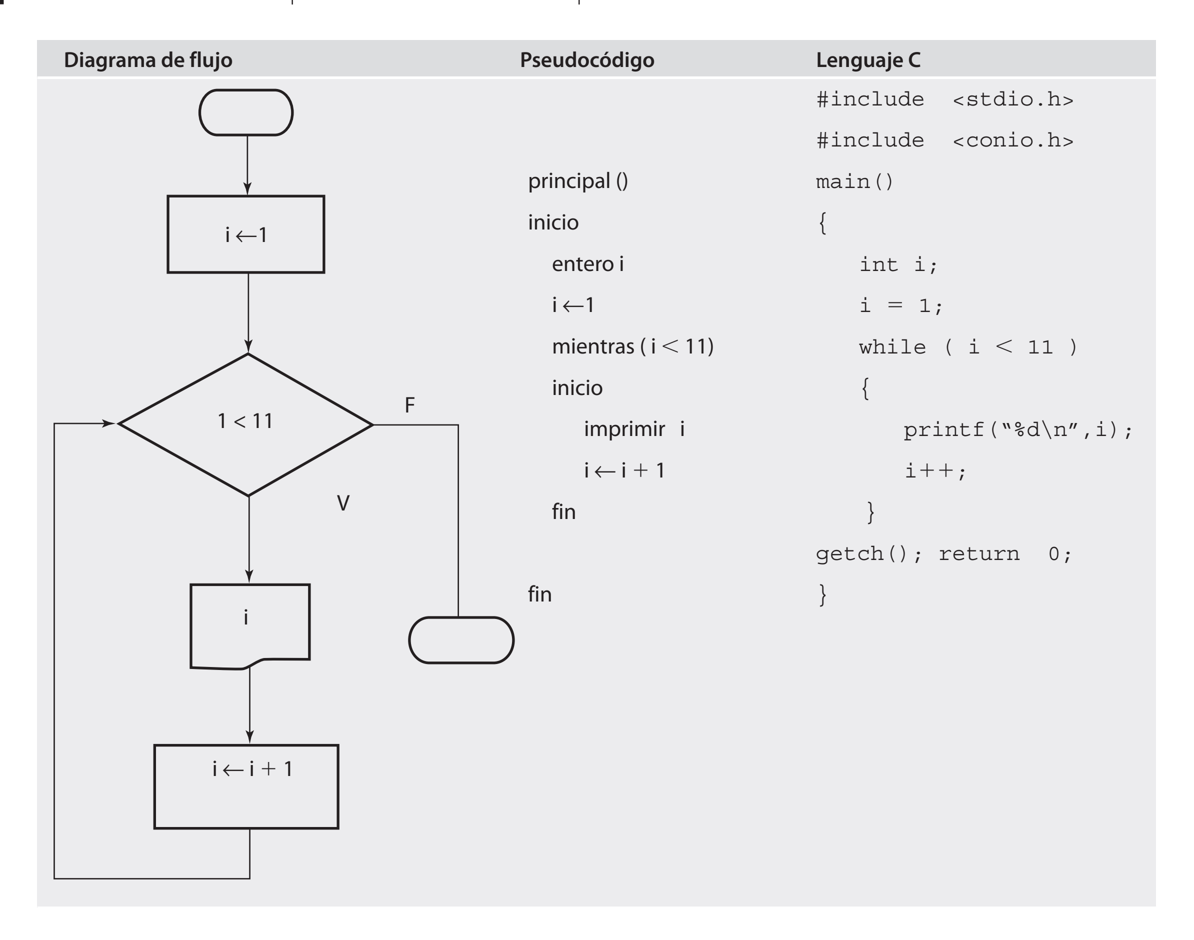
1. La condición (expresión lógica) se evalúa antes del ciclo. Si la condición es verdadera se ejecuta el bucle, y si es falsa se sale y el control pasa a la instrucción siguiente al ciclo.
2. Si la condición es falsa cuando se revisa por primera vez el bucle no se ejecuta nunca, es decir no entra ninguna vez.
3. Mientras la condición sea verdadera el bloque de instrucciones se ejecutará indefinidamente a menos que exista por lo menos una instrucción que modifique el valor de un elemento de la condición.

4. Si existe más de una instrucción se necesitan las palabras reservadas *inicio - fin ({—})* para delimitar el bloque de instrucciones.

Nota: Existen algunos algoritmos y programas que se pueden efectuar con los tres ciclos: *desde*, *mientras* y *hacer_mientras*; aunque cada una tiene sus características específicas.

3.6.3 Ejercicios resueltos de la estructura de control repetitiva mientras (while)

Ejercicio 1. Mostrar los 10 primeros números enteros positivos.




```
principal ()
inicio
   entero i
   i ← 1
   mientras ( i < 11)
   inicio
      imprimir  i
      i ← i + 1
   fin
fin
```

```c
#include  <stdio.h>
#include  <conio.h>
main()
{
   int i;
   i = 1;
   while ( i < 11 )
   {
      printf("%d\n",i);
      i++;
   }
getch(); return  0;
}
```

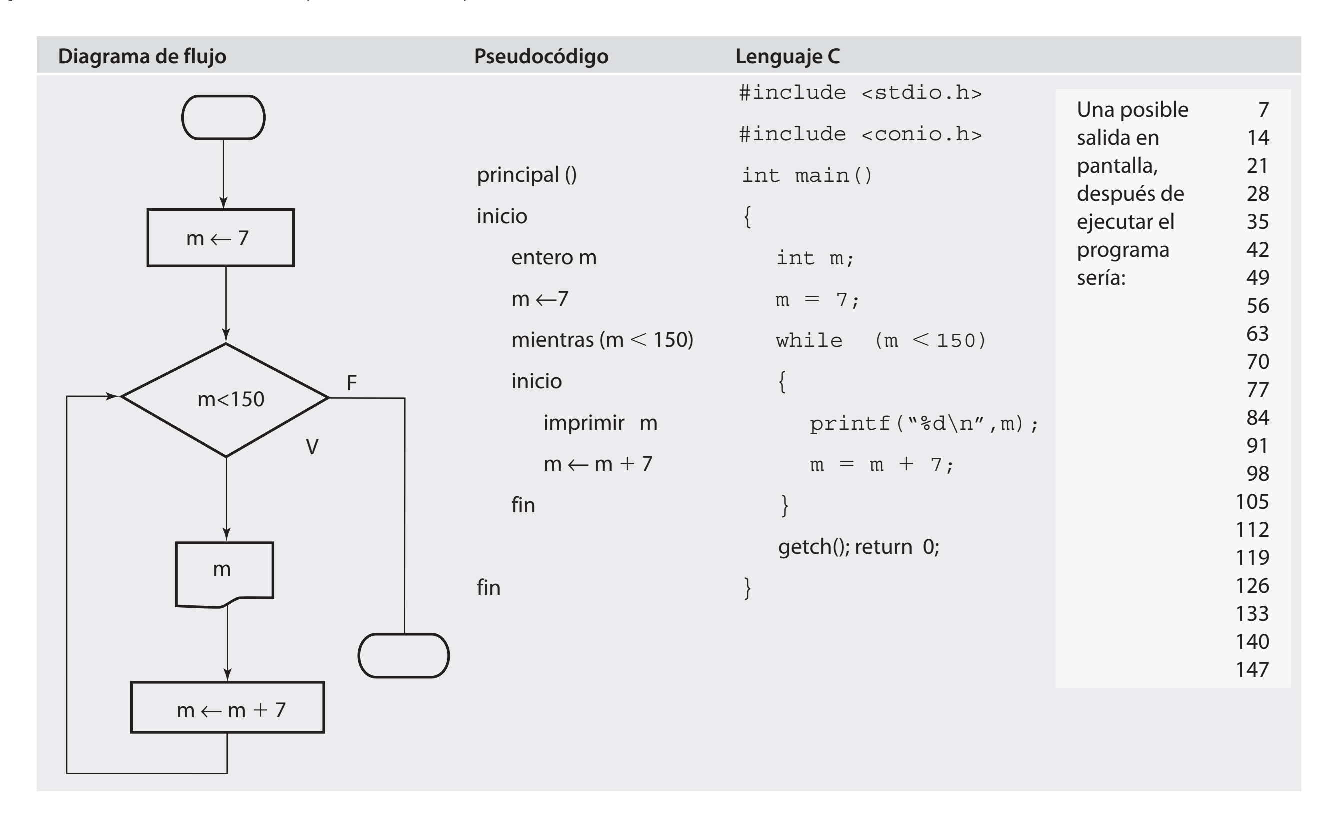
El ejercicio 1 declara la variable *i* de tipo entero, luego se inicializa con el valor de 1 ya que es el primer número entero positivo; se revisa la condición del *mientras (1 < 11);* como la condición se cumple entra al ciclo y se ejecutan las dos instrucciones que están delimitadas en el bloque entre inicio-fin ({—}). Así que imprime el *1* y luego *i←i+1* lo podemos interpretar como la nueva *i* recibe el valor de la i anterior más 1, incrementando en 1 a la variable *i*, de tal forma que $i = 2$, revisa la condición del *mientras* otra vez *(2 < 11)*, como la condición sigue siendo verdadera vuelve a entrar al ciclo, y así sucesivamente hasta que el valor de *i* sea mayor o igual a 11; esto hará que se salga del ciclo y termine el programa. Si vemos la instrucción $i \leftarrow i + 1$, se realiza cada vez que entra al ciclo, esto hace que se cumpla la característica número tres de la estructura repetitiva *mientras*, para que en un momento dado *i* tome un valor mayor que 10 y se salga del ciclo.

El programa imprimirá *1 2 3 4 5 6 7 8 9 10*. El último valor que toma *i* es 11, hace que no se cumpla la condición y por lo tanto se sale del ciclo *mientras*.

Nota: Como se verá más adelante, este ejercicio es igual al ejercicio 1 de la estructura repetitiva "*desde*"; la ventaja principal es que el ciclo *desde* el incremento de su contador *i* lo hace de manera automática, y en la estructura "*mientras*" se debe realizar en una instrucción dentro del ciclo.

Como hemos visto, en lenguaje C después de la condición del *while* no debe escribirse el punto y coma, ya que de hacerlo no se realizará el ciclo; no marca error, pero no aparecerá nada en la pantalla, ya que no entra al ciclo. También podemos apreciar que el equivalente a $i \leftarrow i+1$ es $i++$; aunque también podíamos haber utilizado $++i$.

Ejercicio 2. Mostrar los múltiplos de siete que se encuentren de 0 a 150.



Pseudocódigo

```
principal ()
inicio
    entero m
    m ←7
    mientras (m < 150)
    inicio
        imprimir  m
        m ← m + 7
    fin
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int m;
   m = 7;
   while  (m < 150)
   {
      printf("%d\n",m);
      m = m + 7;
   }
   getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

```
7
14
21
28
35
42
49
56
63
70
77
84
91
98
105
112
119
126
133
140
147
```

En el ejercicio 2 se declara la variable *m* de tipo entero, se inicializa con el valor de 7 ya que es el primer múltiplo de 7 entre 0 y 150; se revisa la condición del *mientras (7 < 150)*, como la condición se cumple entra al ciclo y se ejecutan las dos instrucciones que están delimitadas en el bloque entre inicio-fin ({−}), así que imprime el 7; se le incrementa 7 a la variable *m*, de tal forma que *m = 14*; revisa la condición del *mientras* otra vez *(14 < 150)*; como la condición sigue siendo verdadera vuelve a entrar al ciclo; así sucesivamente hasta que el valor de *m* sea mayor o igual a 150, esto hará que se salga del ciclo y termine el programa.

Si vemos la instrucción *m ← m + 7*, se realiza cada vez que entra al ciclo, esto hace que se cumpla la característica 3, para que en un momento dado *m* tome un valor mayor o igual a 150 y se salga del ciclo.

El programa imprimirá *7 14 21 28 35 42 49 56 63 70 77 84 91 98 105 112 119 126 133 140 147*. El último valor que toma *m* es 154, el cual es mayor a 150 y hace que no se cumpla la condición y se salga del ciclo "*mientras*".

Nota: Cabe mencionar que para cualquier ejercicio donde se maneje más de una instrucción, es importante agrupar el bloque con inicio-fin ({ − }); en caso contrario sólo se realizaría la primera instrucción que se encuentra después del ciclo *mientras (while)*:

Pseudocódigo	Lenguaje C	Resultado
m ← 7	`m = 7`	Se imprimiría indefinidamente el valor de 7 ya que al revisar la condición (7 < 150) ésta siempre será verdadera y volverá a revisarse la misma condición todas las vueltas, ya que nunca se realizaría el incremento m ← m + 7
mientras (m < 150)	`while ( m < 150 )`	
imprimir m	`printf("%d",m);`	
m ← m + 7	`m = m + 7;`	

Como se muestra en este ejercicio, es necesario delimitar la o las instrucciones que queremos que se realicen si la condición es verdadera, ya que el resultado cambia si se le agrega o no el { *y* }, ya que se realizarán una o varias instrucciones.

Ejercicio 3. Imprimir la palabra programación seis veces.

Pseudocódigo	Lenguaje C
Opción A	*Opción A*
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero cont	int cont;
cont ← 1	cont =1;
mientras (cont <= 6)	while (cont <= 6)
inicio	{
imprimir "programación"	printf("programación\n");
cont ← cont + 1	cont = cont + 1;
fin	}
	getch();
fin	}
Opción B	*Opción B*
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero cont	int cont;
cont ← 1	cont =1;
mientras (cont < 7)	while (cont <7)
inicio	{
imprimir "programación"	printf("programación\n");
cont ← cont + 1	cont++ ;
fin	} getch(); return 0;
fin	}

Una posible salida en pantalla, después de ejecutar el programa sería:

programación
programación
programación
programación
programación
programación

El ejercicio 3 está desarrollado de dos formas diferentes, aunque ambas realizan exactamente lo mismo, es decir imprimir seis veces la palabra programación. En lo que difieren es: En la *opción A* la variable *cont* se inicializa en 1 y la condición del *mientras* compara *(cont <= 6)*; al sustituir el valor que compara es *(1 <= 6)* y como la condición se cumple, imprime la palabra programación e incrementa el contador en 1, es decir *cont = 2* y revisa otra vez la condición del *mientras* (2 <= 6); como ésta es verdadera, imprime de nuevo la palabra programación, y así sucesivamente. Los valores que toma la variable *cont* son 1, 2, 3, 4, 5, 6 para que la condición del *mientras* se cumpla y se realice el bloque de instrucciones, pero cuando *cont = 7* la condición ya no se cumple y se sale del ciclo. En la *opción B* la variable *cont* se inicializa con 1 y la condición del *mientras* compara *(cont < 7)*; al sustituir el valor que compara es *(1 < 7)*; como la condición se cumple imprime la palabra programación, y así sucesivamente. Los valores que toma la variable *cont* son 1, 2, 3, 4, 5, 6 para que la condición del *mientras* se cumpla y se realice el bloque de instrucciones, pero cuando *cont = 7* la condición ya no se cumple y se sale del ciclo. En conclusión la programación puede tener diferentes códigos y obtener el mismo resultado. En ambos casos imprime seis veces la palabra programación. Si no se delimita el resultado el bloque será:

Pseudocódigo	Lenguaje C	Resultado
cont ← 1 mientras (cont < 7) imprimir "programación" cont ← cont + 1	`cont = 1;` `while ( cont < 7 )` `printf("programación");` `cont = cont + 1;`	Se imprimiría indefinidamente la palabra programación, ya que al revisar la condición (1 < 7) esta siempre será verdadera y nunca se realizaría el incremento de cont ← cont + 1

Ejercicio 4. Sume un conjunto de números enteros hasta que la suma sea mayor que 100. Al final imprimir el resultado de la acumulación.

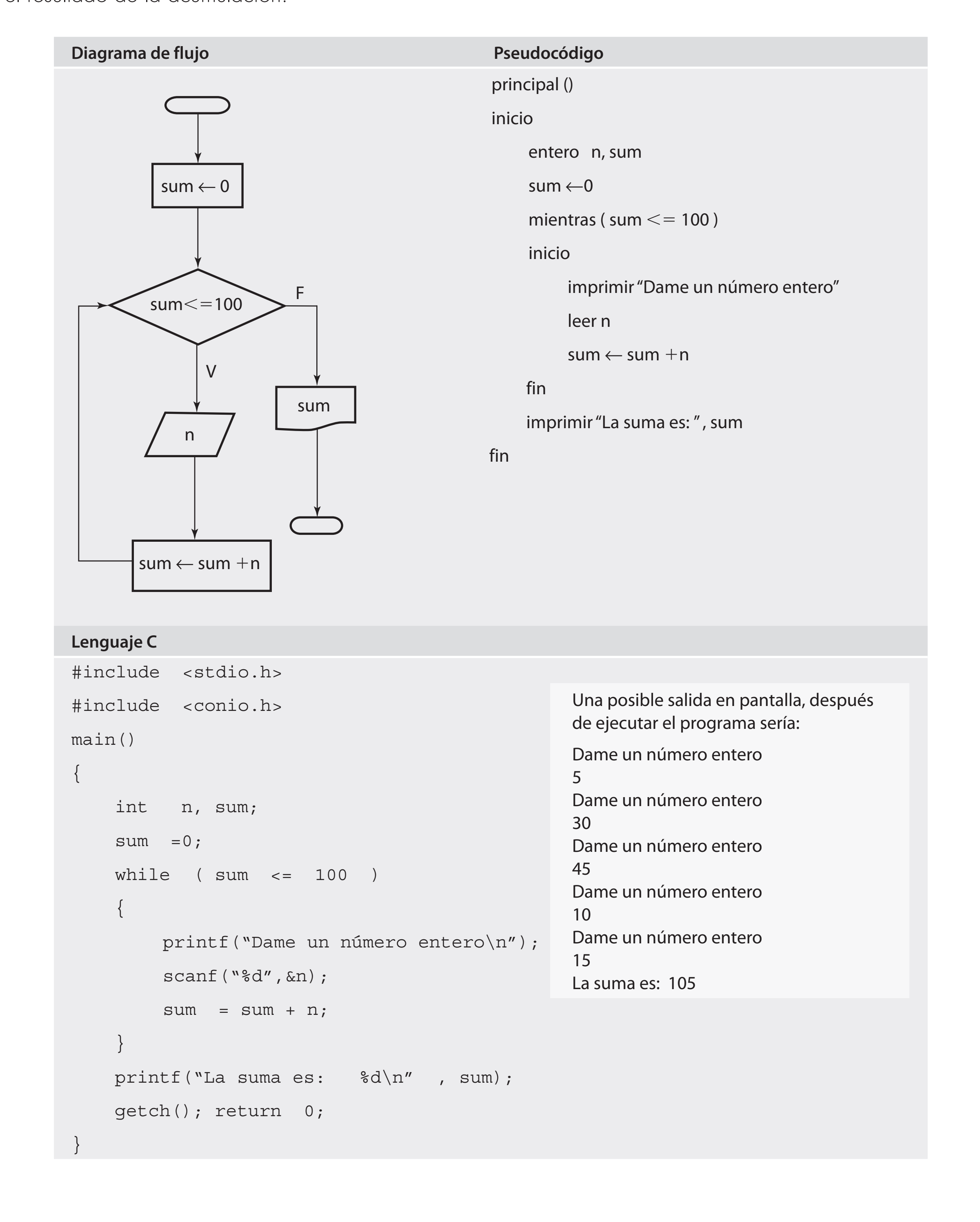


Diagrama de flujo / **Pseudocódigo**

```
principal ()
inicio
    entero  n, sum
    sum ←0
    mientras ( sum <= 100 )
    inicio
        imprimir "Dame un número entero"
        leer n
        sum ← sum +n
    fin
    imprimir "La suma es: ", sum
fin
```

Lenguaje C

```c
#include  <stdio.h>
#include  <conio.h>
main()
{
    int   n, sum;
    sum  =0;
    while  ( sum  <=  100  )
    {
        printf("Dame un número entero\n");
        scanf("%d",&n);
        sum  = sum + n;
    }
    printf("La suma es:   %d\n"  , sum);
    getch(); return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame un número entero
5
Dame un número entero
30
Dame un número entero
45
Dame un número entero
10
Dame un número entero
15
La suma es: 105

El ejercicio 4 difiere respecto a los anteriores, ya que dentro del ciclo *mientras* se va leyendo un número *n*, no se sabe cuántos son, esto dependerá del valor de los números que capture el usuario; si captura números grandes, con algunos la suma llegará a una cantidad mayor que 100, pero si captura números pequeños se necesitará de muchos valores para que la suma llegue a ser mayor que 100. Se declaran dos variables de tipo entero *n* que almacenará los diferentes números y *sum* que irá acumulando la suma de los números que el usuario capture. Se inicializa la variable suma con 0. Se revisa la condición del *mientras* y la primera se cumple *(sum <= 100)* ya que *(0 <= 100)*, así que pregunta por un número y lo almacena en *n*, este número se le suma a la variable *sum* y regresa a revisar la condición del *mientras*; si la suma *(sum)* todavía es menor que 100, vuelve a preguntar por otro número y lo acumula o suma a la variable *sum*, así sucesivamente hasta que la variable *sum* sea mayor que 100.

Ejemplos



En *sum = 0*, se lee el primer número 45, y se almacena en *n*, *sum = 0 + 45*, *sum = 45*, regresa y revisa; la condición es *(45 < 100)*, como ésta se cumple, vuelve a entrar al ciclo *mientras* y lee otro número *n = 30, sum = 45 + 30, sum = 75*, regresa y revisa, la condición es *(75 < 100)*, como también se cumple, vuelve a entrar al ciclo *mientras* y lee otro número *n = 50, sum = 75 + 50, sum = 125*, regresa y revisa la condición, es *(125 < 100)*, como no se cumple, se sale del ciclo y se realiza la siguiente instrucción fuera del ciclo e imprime: *La suma es: 125*. Si no se delimita el resultado el bloque con inicio y fin en pseudocódigo o con {} en lenguaje C quedaría:

Pseudocódigo	Lenguaje C	Resultado
sum ←0 mientras (sum < = 100) imprimir "Dame un número" leer n sum ← sum + n	`sum = 0` `while (sum <= 100)` `printf ("Dame un número");` `scanf("%d",&n);` `sum = sum + n;`	Si la variable sum = 0, se revisa la condición (0 <= 100), esta es verdadera y la única instrucción que se realizará será imprimir "Dame un número". Si otra vez revisa la condición, siempre será verdadera ya que la variable *sum* nunca cambia y por lo tanto se hace cíclica o indefinida.

Ejercicio 5. Calcule el producto (multiplicación) entre un conjunto de números reales hasta que el número introducido sea cero.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real num, prod	`float num,prod;`
prod ← 1	`prod = 1;`
imprimir "Dame un número real "	`printf("Dame un número real\n");`
leer num	`scanf("%f",&num);`
mientras (num <> 0)	`while ( num != 0 )`
inicio	`{`
prod ← prod * num	`prod = prod * num;`
imprimir "Dame otro número"	`printf("Dame otro número\n");`
leer num	`scanf("%f",&num);`
fin	`}`
imprimir "El producto es: ", prod	`printf("El producto es: %f", prod);`
	`getch(); return 0;`
fin	`}`

En el ejercicio 4 introdujimos el concepto sumar o acumular varios números en una sola variable; en este ejercicio se introduce el concepto del producto de varios números. La condición revisa que el número que capturó el usuario sea diferente de 0 ya que si es cero el producto que tengamos se haría 0. Se declaran dos variables reales *num* para almacenar los diferentes números que capture el usuario, y *prod* para que se vayan multiplicando estos números; se inicializa la variable *prod* con 1 para que el primer número que capture se multiplique por la unidad, esta variable se quede con ese valor y a partir de ahí continúe realizando el producto. Se captura un número y se almacena en *num* = *8*, se revisa la condición como *(8 <> 0)* y entra al ciclo prod = prod * num, es decir *prod* = *1 * 8*, *prod* = *8*; se lee otro número y se almacena en num = 12, se revisa la condición como (12 <> 0) y entra al ciclo *prod* = *prod * num*, es decir prod = 8 * 12, prod = 96; se lee otro número y se almacena en num = 2, se revisa la condición como (2 <> 0), entra al ciclo *prod* = *prod * num*, es decir *prod* = *96 * 2*, *prod* = *192*; se lee otro número y se almacena en num. Si el num = 0, se revisa la condición como (0 <> 0), se sale del ciclo y se pasa a la siguiente instrucción que está después del fin del bloque de instrucciones del ciclo "*mientras*", e imprimirá *El producto es: 192.*

Nota: Cabe mencionar que si cualquier ejercicio de los que se han visto no tuviera inicio-fin ({-}) sólo se realizaría la primera instrucción que se encuentra después del ciclo *mientras* (*while*); por ejemplo:

Pseudocódigo	Lenguaje C	Resultado
leer num	scanf("%f",&num);	Si el primer número que se capturó fue diferente de 0, por ejemplo 8; prod = 1 * 8 = 8, y vuelve a revisar la condición del mientras; como el número no cambia y siempre es 8, la condición siempre se cumplirá ya que es 8 <>0 y se hace un ciclo indefinido.
mientras (num <> 0)	while (num != 0)	
prod ← prod * num	prod = prod * num;	
imprimir "Dame otro número"	printf("dame otro número\n");	
leer num	scanf("%f",&num);	

Ejercicio 6. Sumar los números pares y multiplicar los números impares hasta que la suma sea mayor que 50 o el producto mayor que 150.

Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero num, suma, prod	int num, suma, prod ;
suma ← 0	suma = 0;
prod ← 1	prod =1;
mientras (suma <= 50 y prod <= 150)	while (suma <= 50 && prod <= 150)
inicio	{
imprimir "Dame un número entero"	printf ("Dame un número entero \n");
leer num	scanf("%d",&num);
si (num mod 2 = 0)	if (num % 2 = = 0)
suma ← suma + num	suma = suma + num;
si no	else
prod ← prod * num	prod = prod * num;
fin	}
imprimir "La suma es: ", suma	printf("La suma es: %d\n" , suma);
imprimir "El producto es: ", prod	printf("El producto es: %d", prod);
	getch(); return 0;
fin	}

En el ejercicio 6 se revisa si el número es par o impar y, dependiendo de este resultado, lo suma o lo multiplica. Fíjese que las condiciones simples del *mientras* incluyen el operador lógico "y", que servirá para que cuando una de las dos condiciones ya no se cumpla, se salga del ciclo *mientras* y esto satisfaga el objetivo del programa que es cuando alguna de las dos se cumpla y se salga del ciclo. Se inicializan las variables suma=0 y prod=1. Revisa *(suma <= 50 y prod <= 150)*, siendo verdaderas ambas condiciones se entra al ciclo.

Ejemplos



Se lee el primer número *num = 16*, como es par, se *suma = 0 + 16*, quedando un valor *suma = 16*, regresa y se revisa la condición compuesta *(suma <= 50 y prod <= 150)*, como las dos condiciones se cumplen vuelve a entrar al ciclo y se lee otro número *num = 45*, como es impar se multiplica *prod = 1*45, prod = 45*, regresa y se revisan la condición compuesta *(suma <= 50 y prod <= 150)*; como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num = 20*, como es par se suma y queda *suma = 16 + 20, suma = 36*, regresa y se revisa la condición; como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num = 5*, como éste es impar se multiplica *prod = 45 * 5, prod = 225,* regresa y se revisa la condición compuesta *(suma <= 50 y prod <= 150)*; como se cumple sólo la primera condición, se sale del ciclo *mientras* e imprime: *La suma es: 36 El producto es: 225.*

Ejercicio 7. Sumar los números pares y multiplicar los números impares hasta que la suma sea mayor que 50 y el producto mayor que 150.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero num , suma, prod	`int num, suma, prod;`
suma ← 0	`suma = 0;`
prod ← 1	`prod = 1;`
mientras (suma <= 50 o prod <= 150)	`while ( suma <= 50 || prod <= 150 )`
inicio	`{`
imprimir "Dame un número entero"	`printf ("dame un número entero \n");`
leer num	`scanf("%d",&num);`
si (num mod 2 = 0)	`if ( num % 2 = = 0 )`
suma ← suma + num	`suma = suma + num;`
si no	`else`
prod ← prod * num	`prod = prod * num;`
fin	`}`
imprimir "La suma es: ", suma	`printf("La suma es: %d\n",suma);`
imprimir "El producto es: ", prod	`printf("El producto es: %d", prod);`
	`getch(); return 0;`
fin	`}`

Este ejercicio es similar al ejercicio 6. Fíjese que las condiciones simples del *mientras* incluyen el operador lógico "o", que servirá para *mientras* una de las dos condiciones se cumpla vuelva a entrar al ciclo, cuando ninguna de las dos condiciones se cumpla se salga del ciclo *mientras* y esto satisfaga el objetivo del programa, pues ambas condiciones se deben cumplir para salirse del ciclo.

Ejemplos

Se lee el primer número *num* = *36*, como es par, se *suma* = *0* +*36*, y queda un valor *suma* = *36*, regresa y se revisa la condición compuesta; como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num* = *45*, como es impar se multiplica *prod* = *1*45, prod* = *45*, regresa y se revisa la condición compuesta, como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num* = *20*, como es par se suma y queda *suma* = *36* + *20, suma* = *56*, regresa y se revisa la condición compuesta, como se cumple la segunda condición aunque la otra no, vuelve a entrar al ciclo y se lee otro número *num* = *5*; como es impar se multiplica *prod* = *45 * 5, prod* = *225*, regresa y se revisa la condición compuesta, como no se cumplen ninguna de las dos condiciones se sale del ciclo *mientras* e imprime: *La suma es: 56 El producto es: 225*, y cumple de esta manera las especificaciones del programa que la suma sea mayor a 50 y el producto mayor a 150.

Una posible salida en pantalla, después de ejecutar el programa sería: dame un número entero

50

dame un número entero

35

dame un número entero

12

dame un número entero

21

la suma es: 62

el producto es: 735

Ejercicios complementarios de la estructura de control repetitiva *mientras* en pseudocódigo

Pseudocódigos	Contestar las preguntas de cada ejercicio

Ejercicio 1

```
principal ()
inicio
     entero cont
     cont ← 5
     mientras ( cont < 20 )
     inicio
          imprimir "prueba"
          cont ← cont + 2
     fin
fin
```

a) ¿Cuántas veces se imprime la palabra prueba?

____

b) Si no estuvieran las palabras reservadas inicio y fin, ¿cuántas veces imprime la palabra prueba?

____

c) Si el cont se incrementa en tres, es decir cont←cont+3, ¿cuántas veces se imprime la palabra prueba?

____

Ejercicio 2

```
principal ()
inicio
    entero a, b, c
    a ← 3
    b ← 8
    c ← 4
    mientras ( a < 65 )
    inicio
        a ← a * c
        b ← b + a
        c ← c + 1
    fin
    imprimir a,b,c
fin
```

Si realiza una corrida de escritorio con el ejercicio 2. ¿Con qué valores se quedan las variables a, b, c?

a) __________

b) __________

c) __________

Ejercicio 3

```
principal ()
inicio
    entero m
    m ← 3
    mientras ( m < 90 )
    inicio
        imprimir   m
        m ← m + 3
    fin
fin
```

a) ¿Qué realiza el pseudocódigo del ejercicio 3?

____

____

____

b) ¿Con qué valor se queda la variable m? ¿Por qué?

____

____

____

Ejercicio 4

```
principal ()
inicio
    entero i ←1
    mientras ( i < 25 )
    inicio
        imprimir   i
        i ← i + 1
    fin
fin
```

a) ¿Qué realiza el pseudocódigo del ejercicio 4?

____

____

____

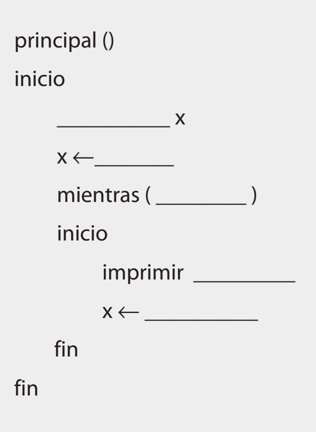
b) ¿Con qué valor se queda la variable i? ¿Por qué?

____

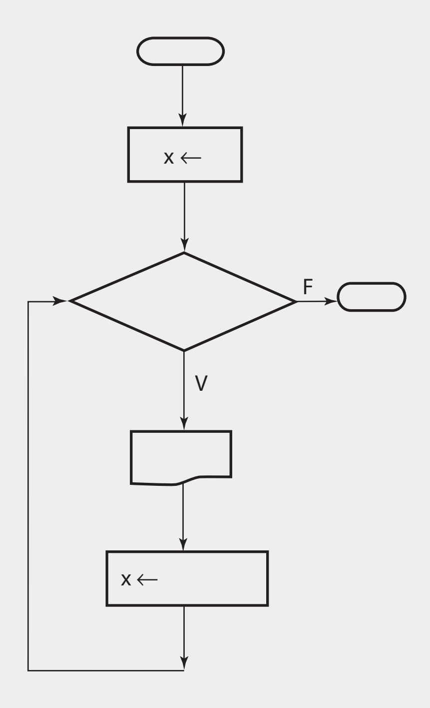
____

____

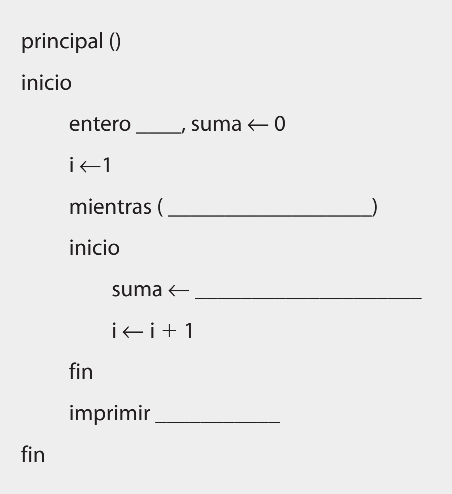
Ejercicio 5. Complete el pseudocódigo y el diagrama de flujo para imprimir todos los múltiplos de 6, entre el 20 y el 200 (incluidos).



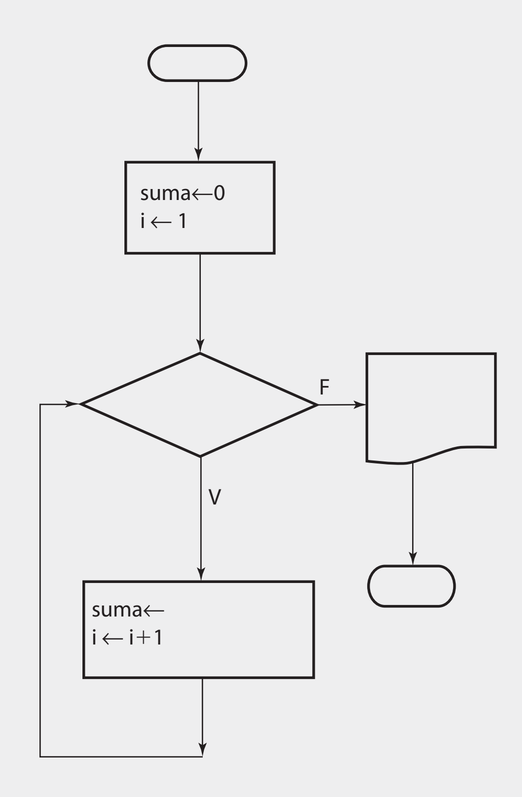
```
principal ()
inicio
    __________ x
    x ←_______
    mientras ( ________ )
    inicio
        imprimir _________
        x ← __________
    fin
fin
```



Ejercicio 6. Complete el pseudocódigo y el diagrama de flujo para sumar los 50 primeros números enteros positivos (1-50). Imprima el resultado.

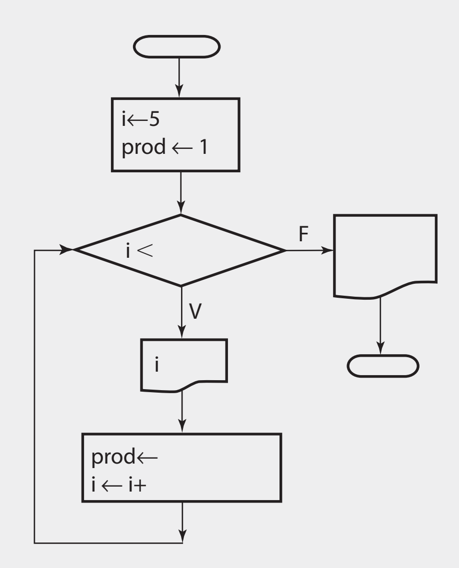


```
principal ()
inicio
    entero ____, suma ← 0
    i ←1
    mientras ( __________________)
    inicio
        suma ← ____________________
        i ← i + 1
    fin
    imprimir ___________
fin
```



Ejercicio 7. Complete el pseudocódigo y el diagrama de flujo que calcule el producto (multiplicación) de los múltiplos de 5, entre 0 y 100 (incluidos). Imprima los múltiplos de 5 y el resultado del producto.

```
principal ()
inicio
    entero i ← 5, prod ← 1
    mientras ( i < _________ )
    inicio
        imprimir i
        prod←_________________
        i ← i + ______
    fin
    imprimir ________
fin
```



Ejercicios complementarios de la estructura repetitiva *while* en lenguaje C

Parte I. Completar el siguiente programa, el cual lee una lista de números y al final imprimir la suma acumulada de los primeros 10 números pares

```
main()
{
   int c=1, acum=0, numero;
   while  (___________)
   {
         printf("Escriba un número entero\n");
         ____________________
         if  (__________________)
             {
                acum=acum+numero; c++;
             }
   }
   printf("Acumulado de los primeros 10 números pares:  %d",acum);
   getch(); return  0;
}
```

Parte II. Elaborar la codificación en lenguaje C de los siguientes programas utilizando la instrucción while (realizando el cálculo e impresión respectivos):

1. Leer una lista de números y calcular el producto acumulado de los primeros 7 múltiplos de 5. Al final imprimir la citada acumulación.

2. Leer una lista de calificaciones y calcular el promedio, el número de aprobados y reprobados; el programa terminará cuando se introduzca un número negativo.
3. El máximo y el mínimo de una lista de números positivos introducidos por teclado, sabiendo que un número negativo termina el ciclo. El negativo no cuenta.
4. Los términos de la serie de Fibonacci hasta que rebase un valor n (entero y positivo) leído por el teclado. En esta serie los dos primeros números son 1, y el resto se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21…
5. Conjetura de ULAM: Empiece con cualquier número entero positivo. Si es par divídalo entre 2 y si es impar multiplique por 3 y aumente en 1. Calcular e imprimir los números enteros positivos repitiendo el proceso hasta llegar a 1. Por ejemplo si empezamos en 5 la serie sería: 5, 16, 8, 4, 2, 1.
6. Calcular las siguientes series:

   a) $1/1 + 1/2 + 1/3 + 1/4 + \ldots 1/n$

   b) $Pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11\ldots$

   c) $1-1/2 + 1/4 - 1/6 + 1/8 - 1/10 + 1/12\ldots$

   El programa debe parar cuando el último término sumado o restado sea menor de 0.01.
7. Leer un número entero e imprimir su equivalente en binario, octal o hexadecimal, según elija el usuario.
8. Conversión de un número arábigo a su respectivo romano. Posteriormente conteste las siguientes preguntas: ¿Qué números romanos aparecen una sola vez? ¿Qué números romanos aparecen varias veces? ¿Qué diferencia existe entre el if y el while?
9. Calcular el factorial de un número entero.
10. Elevar un número real a una potencia entera e imprimir el resultado respectivo.
11. Calcular la media de un conjunto de n número reales.
12. Imprimir de forma inversa los números del 100 al 1, con decremento de uno.
13. Leer un número entero e imprimir si éste es número primo o no.
14. Leer un número (n) mayor que 2 e imprimir los números de la secuencia.

Ejemplos

si n=5: 0+1=1+1=2+1=3+2=5.

Parte III. Realizar la prueba de escritorio o corrida a mano de los siguientes programas:

Ejercicio 1

```
main()
{
   float y=2, x=2,z=3;
   while (z<25||x<10)
   {
       if (z<7) z=z+y;
           else {
                   y=2*y-2;
                   z=z+y;
                   x=y+z;
       }
       y=z/2;
       printf("%f %f %f\n",x,y,z);
   }
   getch(); return  0;
}
```

Ejercicio 2

```
main()
{
    float a=1, b=2,c=3;
       while (b>=0)
       {
          switch (a<c)
          {
            case 0: a=a-b;
                    break;
            case 1: b=c+a;
                    c=b/2;
                    a=a+c;
                    break;
            default: a=c+b*15;
          }
          printf ("%f,%f\n",a,c);
       } getch(); return 0;
}
```

3.6.4 Estructura de control repetitiva hacer_mientras (do while)

Esta estructura le permite al programador especificar que se repita una acción en tanto cierta condición sea verdadera; cuando ésta es falsa se sale del ciclo. La condición la revisa después del ciclo o bucle.

Existen algoritmos y programas que requieren que por lo menos se ejecute el ciclo una vez y al final se revise la condición; en este caso utilizamos la estructura *hacer_mientras*. Es muy semejante al ciclo *mientras*, con la diferencia de que la condición se evalúa después de ejecutar el cuerpo del bucle. Tanto el ciclo *mientras* como el *hacer_mientras* pueden utilizarse cuando no se conoce de antemano el número de veces que se repetirá el ciclo.

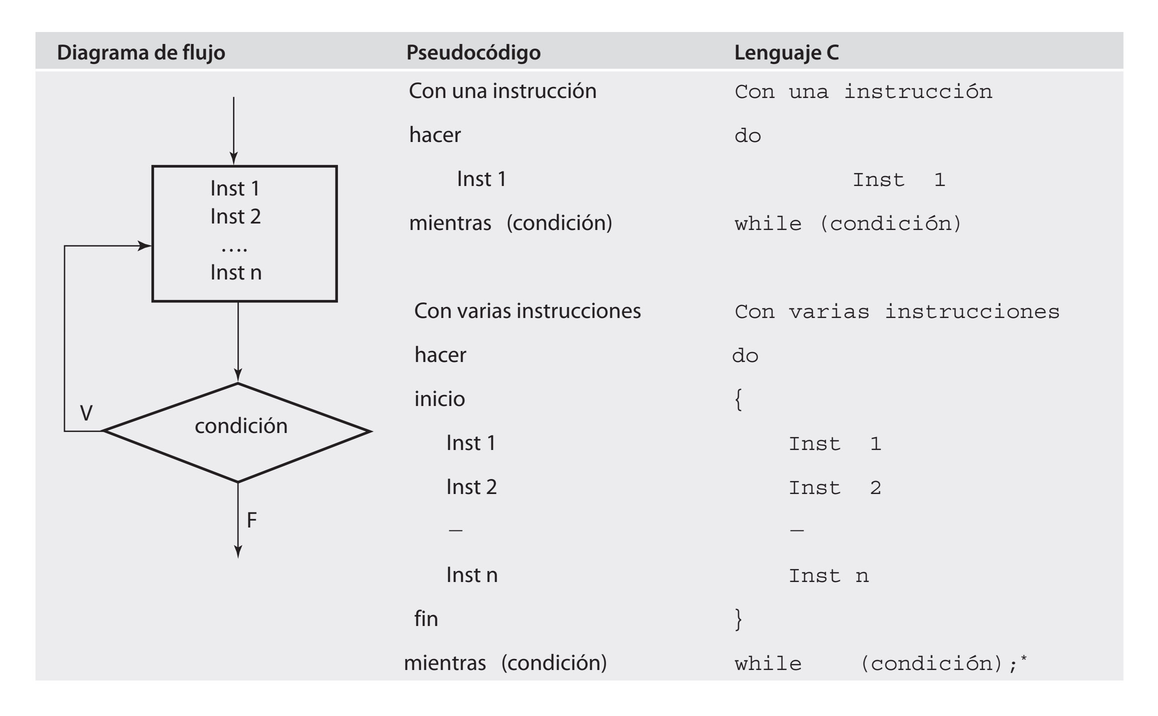


Diagrama de flujo	Pseudocódigo	Lenguaje C
	Con una instrucción	`Con una instrucción`
	hacer	`do`
	Inst 1	`Inst 1`
	mientras (condición)	`while (condición)`
	Con varias instrucciones	`Con varias instrucciones`
	hacer	`do`
	inicio	`{`
	Inst 1	`Inst 1`
	Inst 2	`Inst 2`
	–	–
	Inst n	`Inst n`
	fin	`}`
	mientras (condición)	`while (condición);`*

* A diferencia del while, al final del *do_while* sí lleva punto y coma después de la instrucción *while* (condición). Donde condición es cualquier expresión numérica, relacional o lógica.

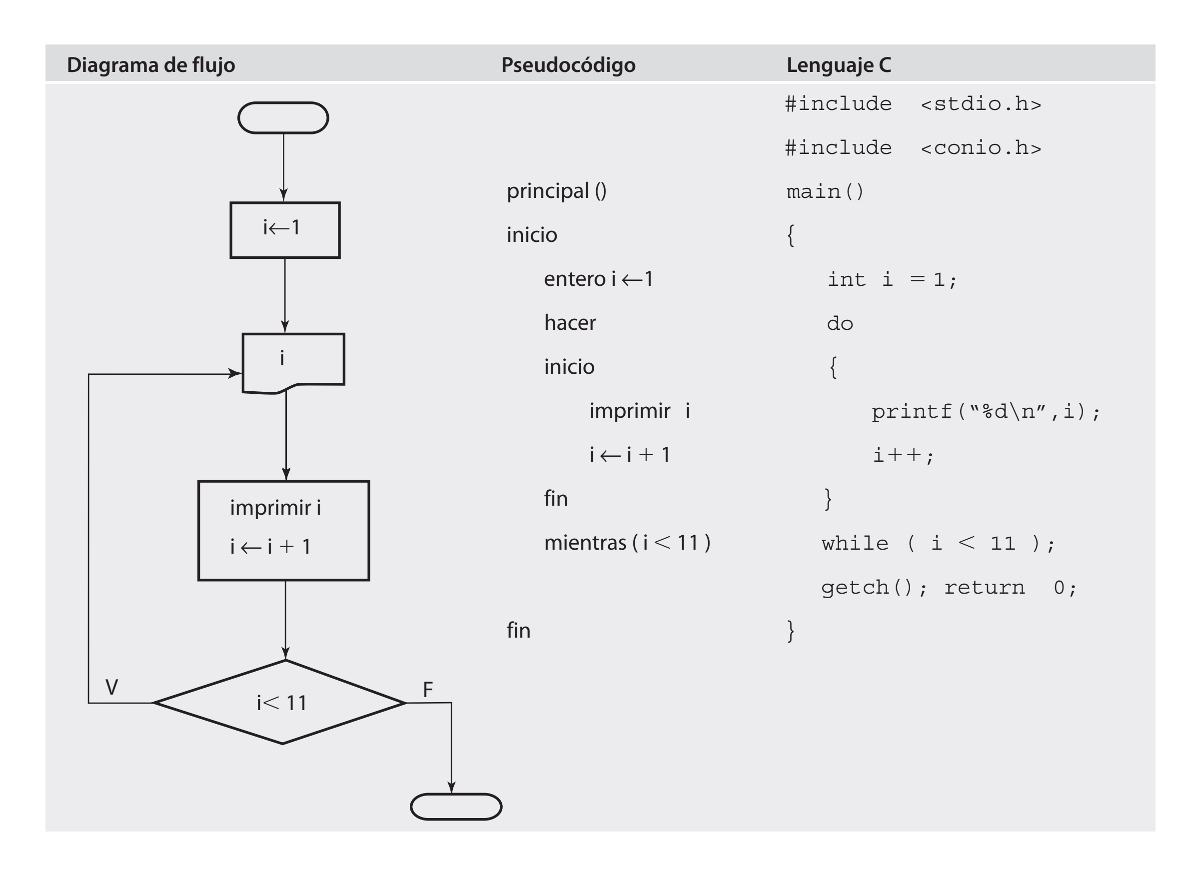
Nota: No es usual que la estructura *hacer_mientras (do_while)* tenga una sola instrucción, generalmente tiene por lo menos dos.

Características:

1. Siempre entra por lo menos una vez al ciclo, ya que la condición está después del ciclo.
2. Si la condición es verdadera entra de nuevo al ciclo y regresa a revisar la condición, hasta que ésta sea falsa se sale del bucle.
3. Debe existir una instrucción dentro del ciclo que modifique la condición, de lo contrario se hace infinita.
4. Si tiene más de una instrucción, necesita obligadamente del inicio-fin ({-}).

3.6.5 Ejercicios resueltos de la estructura de control repetitiva hacer_mientras *(do_while)*

Ejercicio 1. Imprimir los 10 primeros números enteros positivos.



Pseudocódigo

```
principal ()
inicio
    entero i ←1
    hacer
    inicio
        imprimir  i
        i ← i + 1
    fin
    mientras ( i < 11 )
fin
```

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
main()
{
   int i = 1;
   do
   {
      printf("%d\n",i);
      i++;
   }
   while ( i < 11 );
   getch(); return  0;
}
```

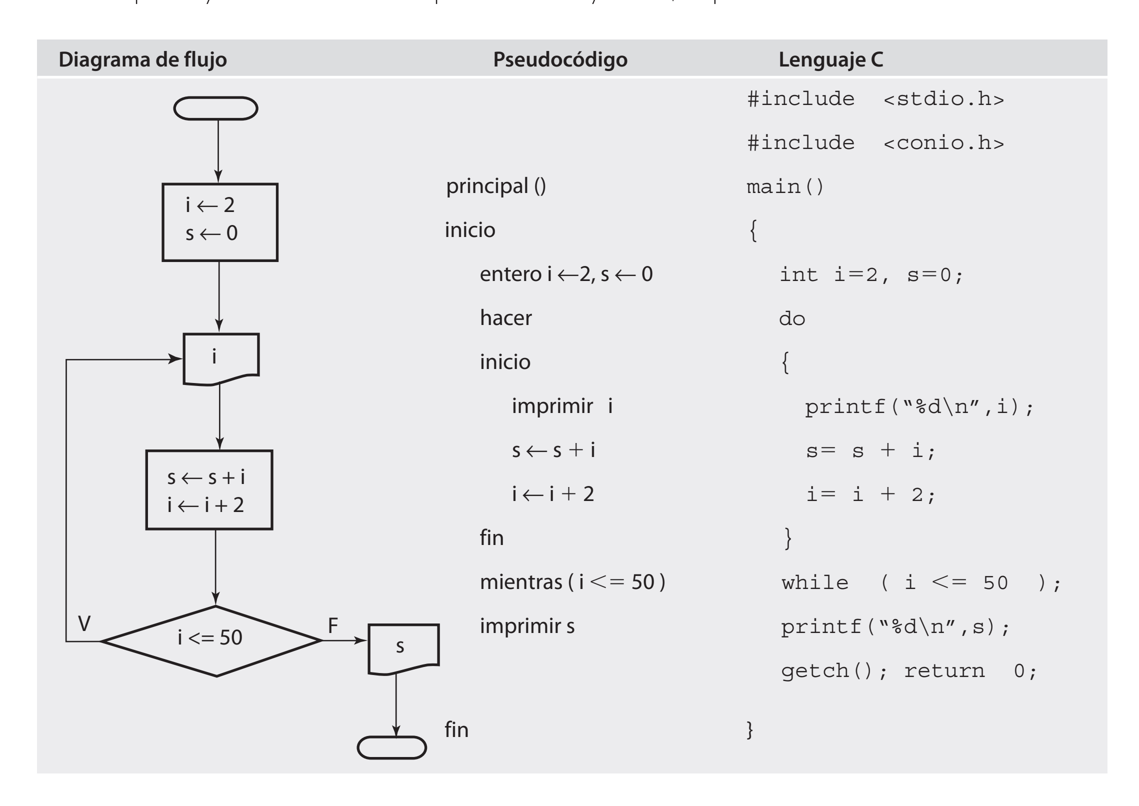
El ejercicio anterior realiza la misma operación que el ejercicio 1 de la estructura *mientras*, pero ahora utilizando la estructura repetitiva *hacer_mientras*. Se declara la variable *i* de tipo entero, se inicializa con el valor de 1 ya que es el primer número entero positivo, se entra al ciclo *hacer_mientras* y se imprime el valor de la variable *i = 1*, se incrementa la variable *i* en 1, es decir *i = 2* y se revisa la condición del *hacer_mientras* (2 < 11). Como la condición se cumple entra al ciclo y se ejecutan las dos instrucciones que están delimitadas en el bloque entre inicio-fin ({-}), se imprime el valor de la variable *i = 2*, se incrementa la variable *i* en 1, es decir *i = 3*, se revisa la condición del *hacer_mientras (3 < 11)*. Como la condición se cumple entra al ciclo y se ejecutan las dos instrucciones que están delimitadas en el bloque entre inicio-fin ({-}), se imprime el valor de la variable *i = 3*, se incrementa la variable *i* en 1, es decir *i = 4* y se revisa la condición del *hacer_mientras (4 < 11)*. Como la condición se cumple entra al ciclo y se ejecutan las dos instrucciones que están delimitadas en el bloque entre inicio-fin ({-}) y así sucesivamente hasta que *i* = 11. Cuando se revisa la condición, como ésta no se cumple se sale del ciclo *hacer_mientras* y termina el programa; esto hace que se cumpla la característica *3* de la estructura repetitiva *hacer_mientras*, para que en un momento dado *i* tome un valor mayor a 10 y se salga del ciclo. El programa imprimirá *1 2 3 4 5 6 7 8 9 10*. El último valor que toma *i* es 11 y por lo tanto no se cumple la condición y se sale del ciclo.

En el ejercicio 2 se inicializa la variable *i* en 2, ya que es el primer número par, y la variable *s* en 0 para guardar la sumatoria. Entra a la estructura *hacer_mientras* y se imprime el 2, se suma el 2 a la variable *s* y queda ésta con un valor de 2, se incrementa *i* en 2 y queda con 4. Se revisa la condición *(4<=50)* y es verdadera, así que regresa a realizar el bloque de instrucciones pero con el valor de *i = 4*, imprime 4, sumando 4 a *s*; ésta queda con 6 y se incrementa nuevamente *i* en 2, que queda con un valor de 6. Realiza repetidamente estas tres instrucciones hasta que *i* sea igual a 50, revisa la condición, es *(50<=50)*, como es verdadera entra por última vez al ciclo repetitivo, imprime 50, sumando 50 al valor que tiene en ese momento s, e incrementa *i* en dos. Queda con un valor de 52, se revisa la condición, que es *(52<=50)*, al ser falsa se sale del ciclo *hacer_mientras* e imprime el valor final de la suma almacenado en *s*.

El programa imprime 2, 4, 6, 8, 10, 12, 14 ....50, y la suma *s* igual a 650.

Nota: En el programa primero entra al ciclo y después revisa si la condición es verdadera, hasta que ésta se hace falsa se sale del ciclo e imprime el valor de la suma *S*.

Ejercicio 2. Imprimir y sumar los números pares entre 1 y el 50, imprimir el resultado de la sumatoria**.**



Pseudocódigo

```
principal ()
inicio
    entero i ←2, s ← 0
    hacer
    inicio
        imprimir  i
        s ← s + i
        i ← i + 2
    fin
    mientras ( i <= 50 )
    imprimir s
fin
```

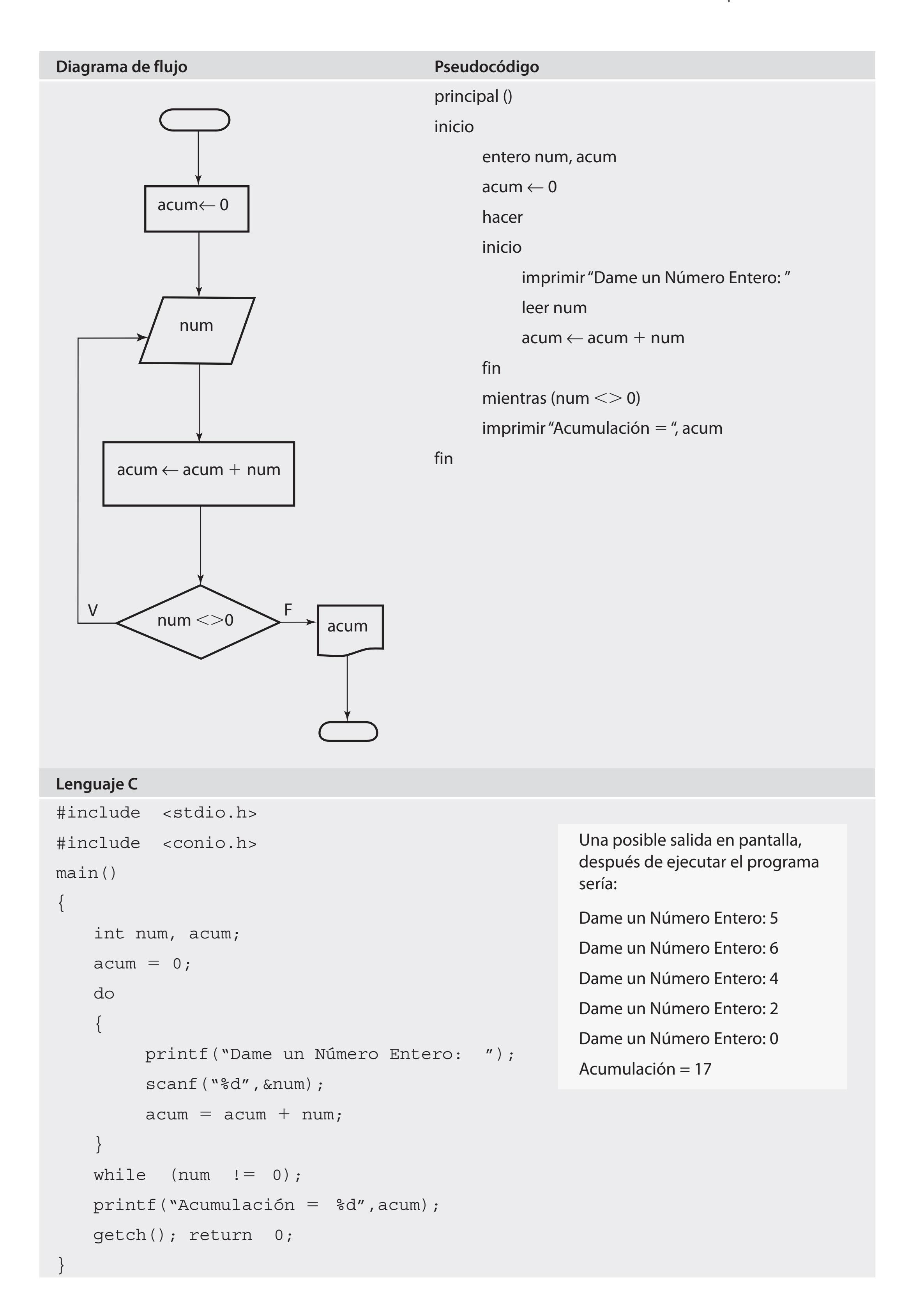
Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
main()
{
   int i=2, s=0;
   do
   {
     printf("%d\n",i);
     s= s + i;
     i= i + 2;
   }
   while  ( i <= 50  );
   printf("%d\n",s);
   getch(); return  0;
}
```

La corrida a mano o prueba de escritorio sería (en cursivas negritas aparecen los valores que se imprimen):

i	*i<=50*	*s = s + i*	*i*	*i<=50*	*s= s + i*
2		0+2	***28***	V	182+28=210
4	V	2+4=6	***30***	V	210+30=240
6	V	6+6=12	***32***	V	240+32=272
8	V	12+8=20	***34***	V	272+34=306
10	V	20+10=30	***36***	V	306+36=342
12	V	30+12=42	***38***	V	342+38 =380
14	V	42+14=56	***40***	V	380+40=420
16	V	56+16=72	***42***	V	420+42=462
18	V	72+18=90	***44***	V	462+44=506
20	V	90+20=110	***46***	V	506+46=552
22	V	110+22=132	***48***	V	552+48=600
24	V	132+24=156	***50***	V	600+50=650
26	V	156+26=182	***52***	F	

Ejercicio 3. Calcule la sumatoria de un conjunto de números, mientras el número que se capture en cada ciclo sea diferente de 0. Al final imprimir el resultado respectivo.



Pseudocódigo

```
principal ()
inicio
    entero num, acum
    acum ← 0
    hacer
    inicio
        imprimir "Dame un Número Entero: "
        leer num
        acum ← acum + num
    fin
    mientras (num <> 0)
    imprimir "Acumulación = ", acum
fin
```

Lenguaje C

```
#include  <stdio.h>
#include  <conio.h>
main()
{
   int num, acum;
   acum = 0;
   do
   {
        printf("Dame un Número Entero:  ");
        scanf("%d",&num);
        acum = acum + num;
   }
   while  (num  !=  0);
   printf("Acumulación =  %d",acum);
   getch(); return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame un Número Entero: 5
Dame un Número Entero: 6
Dame un Número Entero: 4
Dame un Número Entero: 2
Dame un Número Entero: 0
Acumulación = 17

En el ejercicio 3 la variable *acum* es un acumulador, el cual cumple con las dos características antes mencionadas: se inicializó con 0 *(acum ← 0)* antes del entrar al ciclo *hacer_mientras* y la variable aparece a los dos lados de la asignación dentro del ciclo *(acum ← acum + num)*. En cada vuelta del ciclo se lee un número *(num)* y se va acumulando, por lo tanto se realiza la sumatoria de los números capturados hasta que se capture el

número cero, ya que la condición se hará falsa y se rompe el ciclo. En el ejemplo del recuadro capturamos diferentes valores para *num* comenzando con 5, dicho número se suma al 0, como el 5 es diferente de 0, la condición es verdadera y da otra vuelta y así sucesivamente repetirá el proceso con los demás números hasta que se capture el 0 y obtenga la acumulación final, y por lo tanto imprime: *Acumulación = 17*.

Ejercicio 4. Introducir un número entero y contar cuántos dígitos tiene.

En el ejercicio 4 la variable contador *c* se inicializa con 0, leemos un número entero *num*, el cual asignamos a la variable *num1*; para no perder el valor de *num*, dicha variable en cada vuelta es dividida entre 10 tantas veces hasta que desaparece, esto debido a que la numeración que utilizamos es base 10.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero num, num1, c	`int num, num1, c;`
c ← 0	`c = 0;`
imprimir "Dame un Número Entero: "	`printf("Dame un Número Entero:");`
leer num	`scanf ("%d", &num);`
num1 ← num	`num1 = num;`
hacer	`do`
inicio	`{`
num ← num / 10	`num = num / 10;`
c ← c + 1	`c++;`
fin	`}`
mientras (num >= 1)	`while (num >= 1);`
imprimir "El Número tiene ", c,"Dígitos"	`printf("El Número tiene %d Dígitos",c);`
	`getch(); return 0;`
fin	`}`

En el ejemplo capturamos el número 1234, en la primera vuelta el número se divide entre *10* y *num=123* ya que sólo almacena la parte entera; el contador *c = 1*, la condición *(num>=1)* es decir *(123>=1)* es verdadera; en la siguiente vuelta dividimos *123* entre *10* y *num=12* y *c = 2*; la condición *(12>=1)* es verdadera. Seguirá dando vueltas hasta que *num* se hace 0, en ese momento la condición *(num>=1)* es falsa y por lo tanto se rompe el ciclo. Imprime: *El Número Tiene 4 Dígitos*. La corrida a mano o prueba de escritorio es:

c = 0	num ← 1234	
Número de ciclo	*num ← num / 10*	*c ← c + 1*
1	1234/10 = 123	0 + 1 = 1
2	123/10 = 12	1 + 1 = 2
3	12/10 = 1	2 +1 = 3
4	1/10 = 0	3 +1 = 4

Ejercicio 5. Adivinar en un máximo de cinco oportunidades un número entero comprendido entre 1 y 100. En cada ciclo la computadora dirá si el número que capturó el usuario es mayor o menor que el número de la computadora. Al final si adivinó el número, imprimir: "Felicidades lo lograste en ____ intentos", y si no imprimir "Lástima, suerte para la próxima".

Pseudocódigo

```
constante NMAQ ← 17
principal ()
inicio
    entero c, nusuario
    c ← 0
    hacer
    inicio
        imprimir "Dame un Número Entero";   leer nusuario
        si (nusuario > NMAQ)
            imprimir "Tu número es Mayor al Mío"
        si (nusuario < NMAQ)
            imprimir "Tu número es Menor al Mío"
        c ← c +1
    fin
    mientras (nusuario <> NMAQ y c < 5)
    si (nusuario = NMAQ)
        imprimir "Felicidades, lo lograste en ", c, "Intentos"
    si no
        imprimir "Lástima, suerte para la próxima"
fin
```

En el ejercicio 5 se utiliza la constante *NMAQ*, que sirve para que el programador pueda capturar un número y éste pueda ser adivinado por el usuario; la variable *c* servirá de contador para las cinco oportunidades y la variable *nusuario* para almacenar los números que el usuario vaya introduciendo para adivinar el correcto; la variable *c* se inicializa con 0. Por ejemplo, si el primer número que el usuario captura es *nusuario=10* se compara con el valor de la constante *NMAQ*, como éste es menor se imprime *Tu número es Menor al Mío*, se incrementa la variable *c* en *1*, quedando con valor 1, y se revisa la condición mientras *(10<>17 y 1<5)*; como ambas condiciones se cumplen se repiten las instrucciones del ciclo, si el segundo número que el usuario captura es *nusuario=18* se compara con el valor de la constante *NMAQ*, como éste es mayor se imprime *Tu número es Mayor al Mío*, se incrementa la variable *c* en 1 y queda con valor 2, se revisa la condición compuesta mientras *(18<>17 y 2<5)*, como ambas condiciones se cumplen se repiten las instrucciones del ciclo; si el tercer número que el usuario captura es *nusuario=17* se compara con el valor de la constante *NMAQ*, como el número no es menor ni tampoco mayor, no imprime nada pero sí se incrementa la variable *c* en 1 y queda con valor 3, se revisa la condición compuesta *mientras (17<>17 y 3<5)*; la primera condición no se cumple ya que son iguales pero la segunda condición sí; por lo tanto la condición compuesta es falsa (ya que están utilizando el operador lógico y donde ambas se tienen que cumplir para que sea verdadero), se sale del ciclo *hacer_mientras* e imprime *Felicidades lo lograste en 3 Intentos*. Si después de cinco intentos no logra adivinar el valor, el valor de la variable c = 6 y la condición compuesta será: *(4<>17 y 6<5)*, donde se cumple la primera condición pero no la segunda; la condición compuesta es falsa y se sale del ciclo *hacer_mientras*.

si *nusuario=4*, revisa la condición *(nusuario = NMAQ)*, donde *(4 = 17)*, la cual es falsa, el programa imprime *Lástima, suerte para la próxima*.

Ejemplos



Nota: Es recomendable utilizar letras mayúsculas como identificadores de constantes y minúsculas como identificadores de las variables.

A diferencia del pseudocódigo, en lenguaje C utilizamos las funciones srand() y rand() para generar números aleatorios.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
main()
{
   int c, nusuario,nmaq;
   c = 0;
   srand(time(NULL)); nmaq = rand()%100 + 1;
   do
   {
        printf("dame un número entero\n");
        scanf("%d",&nusuario);
        if (nusuario > nmaq) printf("Tu número es mayor al mío \n");
        if (nusuario > nmaq) printf("Tu número es menor al mío \n");
        c = c + 1;
   }
   while (nusuario != nmaq && c < 5);
   if  (nusuario == nmaq) printf("Felicidades lo lograste en %d
intentos",c);
              else printf("Lástima, suerte para la próxima");
   getch(); return  0;
}
```

Nota: Vea el ejemplo anterior en turbo C, en el capítulo de programación modular.

La función rand()

En C, para generar números aleatorios utilizamos la función *rand();* cada vez que la llamamos nos devuelve un número entero aleatorio entre *0* y el *RAND_MAX*. La constante RAND_MAX está definida en la librería stdlib.h. Veamos un ejemplo:

Ejemplos



```
#include  <stdlib.h>
...
numero  = rand() % 26;  //número aleatorio entre 0 y 25
numero  = rand() % (N + 1);  //número aleatorio entre 0 y N
```

La operación módulo (%) nos da el resto de dividir *rand()* entre *26*. Este resto puede ir de *0* a *25*. De la misma forma, el módulo de *rand()* entre *N+1* va de *0* a *N*.

De manera general si el rango está entre *25* y *100* (donde *D* es mayor que *I*). Primero generamos un número entre *0* y *75* y le sumamos *25* (un número entre *0* y *D−I* y le sumamos *I*:

numero = rand () % 76 + 25; // Valores entre 25 y 75
numero = rand () % (D −I +1) + I; // Este está entre I y D

La función srand()

Si ejecutamos varias veces nuestro programa, la secuencia de números aleatorios se repite. Para evitar este problema tenemos la función *srand()*, a la que se le pasa de parámetro un número que se utilizará como número inicial. En nuestro ejemplo utilizamos el reloj de la computadora con la función *time()*. Sólo debemos llamarla una vez en nuestro programa.

Ejercicio 6. Inicializar un contador con 25, imprimir el letrero programación hasta que dicho contador sea menor que 12; el contador se irá decrementando de tres en tres.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero c ← 25	`int c = 25;`
hacer	`do`
inicio	`{`
imprimir "PROGRAMACIÓN"	`printf("PROGRAMACIÓN\n");`
c ← c – 3	`c =c − 3;`
fin	`}`
mientras (c > 12)	`while (c > 12);`
	`getch(); return 0;`
fin	`}`

El ejercicio 6 utiliza una variable *c*, la cual se inicializa en *25*, entra al ciclo *hacer_mientras* e imprime la palabra PROGRAMACIÓN, se decrementa en tres a la variable y queda *c= 22*, se revisa la condición del *mientras (c>12)*, es decir *(22>12)*. Como la condición se cumple se repite el bloque de instrucciones que se encuentran delimitadas entre el inicio y fin del *hacer_mientras*, entra al ciclo *hacer_mientras* e imprime la palabra PROGRAMACIÓN, se decrementa en tres a la variable y queda *c= 19*, revisa la condición del mientras *(c>12)*, es decir *(19>12)*. Como la condición se cumple se repite el bloque de instrucciones que se encuentran delimitadas entre el inicio y fin del *hacer_mientras*, entra al ciclo *hacer_mientras* e imprime la palabra PROGRAMACIÓN, decrementa en tres a la variable y queda *c=16*, revisa la condición del *mientras (c>12)*, es decir *(16>12)*. Como la condición se cumple se repite el bloque de instrucciones que se encuentran delimitadas entre el inicio y fin del *hacer_mientras*, entra al ciclo *hacer_mientras* e imprime la palabra PROGRAMACIÓN, decrementa en tres a la variable y queda *c=13*, revisa la condición del *mientras (c>12)*, es decir *(13>12)*. Como la condición se cumple se repite el bloque de instrucciones que se encuentran delimitadas entre el inicio y fin del *hacer_mientras*, entra al ciclo *hacer_mientras* e imprime la palabra PROGRAMACIÓN, se decrementa en tres a la variable quedando *c= 10*, se revisa la condición del *mientras (c>12)*, es decir *(10>12)*. Como la condición no se cumple se sale del ciclo *hacer_mientras* y termina el programa, es decir, imprime cinco veces la palabra PROGRAMACIÓN.

Ejercicio 7. Sumar los números pares y multiplicar los números impares hasta que la suma sea mayor que 50 o el producto mayor que 150.

Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero num, suma ← 0, prod ← 1	int num, suma = 0,prod = 1;
hacer	do
inicio	{
imprimir "Dame un número entero"	printf("dame un número entero \n");
leer num	scanf("%d",&num);
si (num mod 2 = 0)	if (num % 2 == 0)
suma ← suma + num;	suma ← suma + num;
si no	else
prod ← prod * num	prod = prod * num;
fin	}
mientras (suma <= 50 y prod <= 150)	while (suma <= 50 && prod <= 150);
imprimir "La suma es: ", suma	printf("la suma es: %d\n", suma);
imprimir "El producto es: ", prod	printf("el producto es: %d\n",prod);
	getch(); return 0;
fin	}

Este ejercicio es igual que el ejercicio 6 de la estructura *mientras*, sólo que en este ejercicio del *hacer_mientras* se revisa si el número es par o impar y, dependiendo de este resultado, lo suma o lo multiplica; fíjese que la condición compuesta del *mientras* incluye el operador lógico "y", que servirá para que cuando una de las dos condiciones ya no se cumpla se salga del ciclo *mientras* y esto satisfaga el objetivo del programa que es cuando alguna de las dos se cumpla y se salga del ciclo.

Ejemplos



Se lee el primer número *num* = *16*, como es par, se *suma* = *0* +*16*, queda un valor *suma* = *16*, y se revisa la condición compuesta (*16* <=*50* y *1*<=*150*). Como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num* = *45*, como es impar se multiplica *prod* = *1*45*, *prod*=*45* y se revisa la condición compuesta, como se cumplen las dos condiciones de *(16*<=*50* y *45*<=*150)*, vuelve a entrar al ciclo y se lee otro número *num* = *20*, como es par se suma y queda *suma* = *16* + *20*, *suma* = *36*, se revisa la condición compuesta *(36*<=*50* y *45*<=*150*). Como se cumplen las dos condiciones vuelve a entrar al ciclo y se lee otro número *num* = *5*, como es impar se multiplica *prod* = *45*5*, *prod*=*225*, se revisa la condición compuesta *(36*<= *50* y *225*<=*150*), como se cumple una condición *(36*<=*50)* y otra no *(225*<=150) se sale del ciclo *hacer_mientras* e imprime: *La suma es: 36 El producto es: 225*.

Ejercicios complementarios de la estructura de control repetitiva *hacer_mientras* en pseudocódigo

Pseudocódigos	**Contestar las preguntas de cada ejercicio**

Ejercicio 1

```
principal ()
inicio
    entero cont ← 3
    hacer
    inicio
        imprimir "Examen"
        cont ← cont + 4
    fin
    mientras ( cont < 25 )
fin
```

a) ¿Cuántas veces se imprime la palabra Examen?

____

b) Si no estuvieran las palabras reservadas inicio y fin ¿cuántas veces imprime la palabra Examen?

____

c) Si fuera cont=cont +3 ¿cuántas veces imprime la palabra Examen?

____

Ejercicio 2

```
principal ()
inicio
    __________ m ←_______
    hacer
    inicio
        imprimir
        m ← __________
    fin
    mientras ( ________ )
fin
```

a) Completar el programa para que se impriman todos los múltiplos de 8, entre el 10 y el 250

____

b) ¿Con qué valor se queda al final la variable m? ¿Por qué?

____

____

____

____

Ejercicio 3

```
principal ()
inicio
    entero a ← 3, b ← 7
    hacer
    inicio
        a ← a * b
        b ← b + a
    fin
    mientras ( a < 70 )
    imprimir a, b
fin
```

Si realizas una corrida de escritorio con el ejercicio 3, ¿con qué valores se quedan las variables a, b?

a ______________

b ______________

Ejercicio 4

```
principal ()
inicio
    entero m
    m ←5
    hacer
    inicio
        imprimir  m
        m ← m + 5
    fin
    mientras ( m < 100 )
fin
```

a) ¿Qué realiza el pseudocódigo del ejercicio 4?

____

____

____

b) ¿Con qué valor se queda la variable m? ¿Por qué?

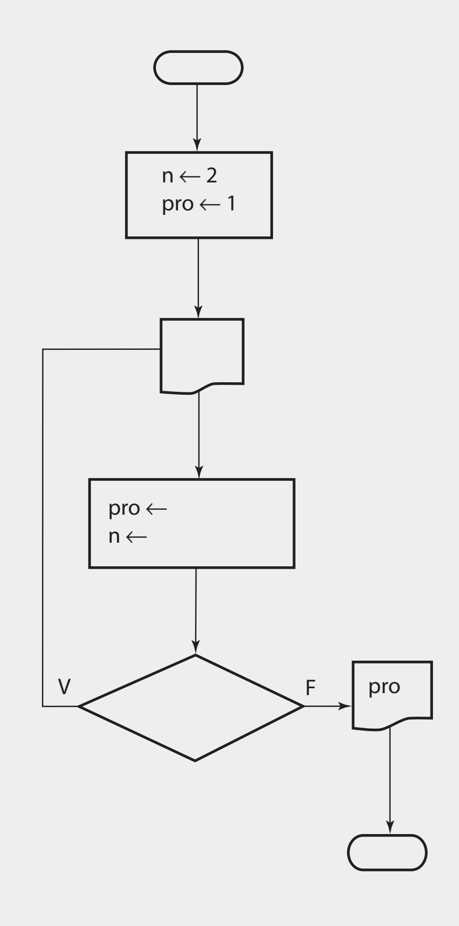
____

____

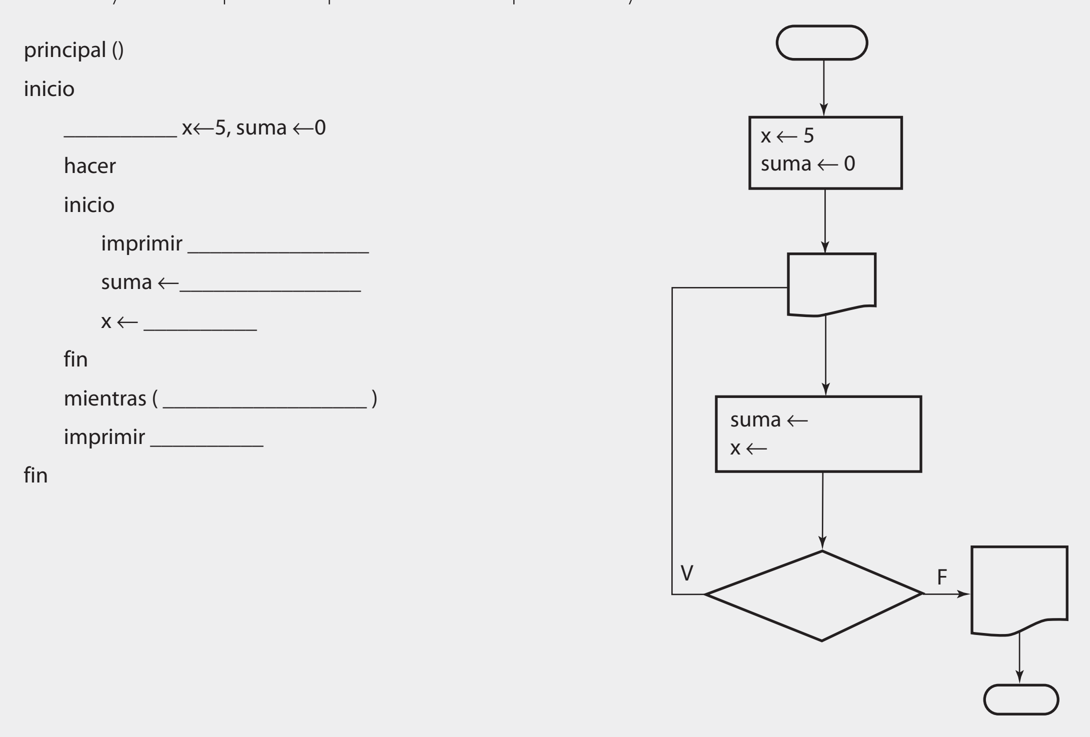
____

Ejercicio 5. Complete el pseudocódigo y el diagrama de flujo para imprimir y multiplicar los 50 primeros números pares. Imprimir el producto final.

```
principal ()
inicio
    __________ n←2 , pro ← 1
    hacer
    inicio
        imprimir
        pro ← __________
        n ← __________
    fin
    mientras ( ___________________ )
    imprimir "producto =", pro
fin
```

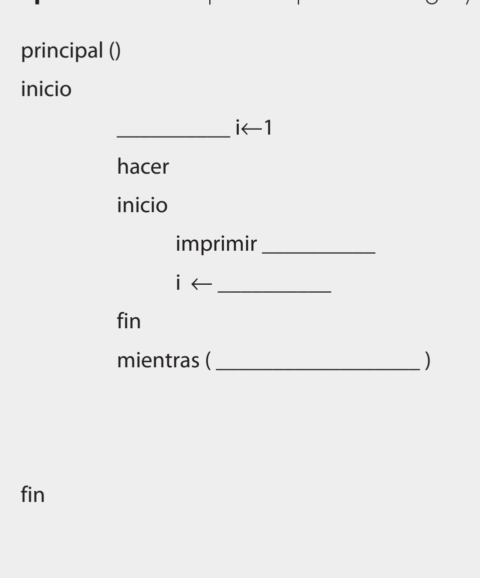


Ejercicio 6. Complete el pseudocódigo y el diagrama de flujo que calcule la suma de los múltiplos de 5, entre 0 y 100. Imprimir en pantalla los múltiplos de 5 y el resultado de la suma.



```
principal ()
inicio
      __________ x←5, suma ←0
      hacer
      inicio
            imprimir ________________
            suma ←________________
            x ← __________
      fin
      mientras ( __________________ )
      imprimir __________
fin
```

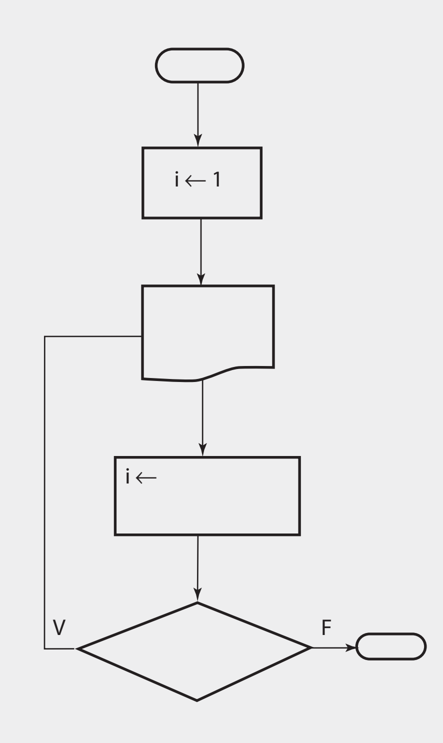

Ejercicio 7. Complete el pseudocódigo y el diagrama de flujo que imprima 15 veces la palabra “Adiós”.



```
principal ()
inicio
            __________ i←1
            hacer
            inicio
                  imprimir __________
                  i ← __________
            fin
            mientras ( __________________ )



fin
```



Ejercicios complementarios de la estructura de control repetitiva *do-while* en lenguaje C

Parte I. Realizar la prueba de escritorio o corrida a mano de los siguientes programas

Ejercicio 1

```
main()
{
    int x = 0;
    do
    {
        x = x + 1;
        x = x * x;
        printf("%d\n",x);
    }
    while (x <= 55);
    getch(); return  0;

}
```

Ejercicio 2

```
main()
{
  int a=1, b=2, c=2;
       do
       {
          if(b%3!=0)c=c*2;
             else{
                  a=b+c;
                  c=a*(c−b);
                  }
                 b=b+a;
                 printf("%d,%d\n",b,c);
       }while (c<50 || b<20);
  getch();
}
```

Ejercicio 3

```
main()
{
   int i=2, j=1, k=10;
   printf  ("%d,  %d\n", i, k/2+i*2);
   do
   {
       if (k<=10)
       {
          j=50−i*2;   i+=2;
       }
       k+=i;      i+=10;
       printf ("%d, %d\n",i, k/2+i*2);
   }
   while (i<20 &&j>45); getch(); return 0;
}
```

Ejercicio 4

```
main()
{
   int a=10,    b=10, c=4;
   do
   {
       if  (a>5)
        {
           a/=2; b=c+2;
       }
       else a=c*2;
           c=b+a%5;
       printf("%d,%d\n",a,c);
   }
   while (a<30&&c<100); getch(); return 0;
}
```

Parte II. Elaborar la codificación en lenguaje C de los siguientes programas utilizando la instrucción do_while (realizando el cálculo e impresión respectiva):

1. Imprimir los cuadrados de los 25 primeros números enteros divisibles entre 7.
2. Leer un número entero de 3 dígitos (validar la captura respectiva, sólo valores entre 1 y 999) y separarlo en unidades, decenas y centenas.
3. Simular una ruleta (números aleatorios entre 0 y 38). Dejará de girar cuando se genere el cero. En cada vuelta contará por separado pares y nones. Al final imprimir los resultados.
4. Las potencias de 2 son: 1, 2, 4, 8, 16,... Para obtener la siguiente potencia se multiplica la anterior por 2. Imprimir la primera potencia de 2 que pasa de 700.
5. Leer dos números enteros y calcular su producto con el algoritmo ruso. Para calcular el producto de 45 y 15 se obtendrá: 22-30, 2-240, Producto= 675.
6. De acuerdo con un principio aritmético un número es múltiplo de 3 cuando la suma de sus cifras da un múltiplo de 3. Con base en este principio imprimir si un número es múltiplo de 3.
7. Leer un número e imprimir cuántos dígitos son impares y cuántos son pares.



Si N= 26455, imprimirá: el número tiene 3 dígitos pares y 2 impares.

8. Calcular el M.C.D y el M.C.M. de dos números leídos.
9. Generar dos números aleatorios entre 0 y 99, imprimir dichos números en forma de resta (por ejemplo: 25 − 10 =), y leer el resultado respectivo. En caso de ser correcto imprimir “Felicidades, tu resultado es correcto”, en caso contrario imprimir “Lo siento, el resultado era:___”. El proceso se llevará a cabo hasta que el alumno conteste correctamente.
10. Calcular el factorial de un número entero.
11. Elevar un número real a una potencia entera.
12. Calcular la media de un conjunto de *n* número reales.
13. Imprimir de forma inversa los números del 100 al 1, con decremento de uno.
14. Leer un número entero e imprimir si éste es número primo o no.

3.6.6 Estructura de control repetitiva desde (for)

El *desde (for)* es la estructura repetitiva más utilizada y simple de manejar, ya que repite un conjunto de instrucciones un número determinado de veces. Una de sus aplicaciones principales son los arreglos.

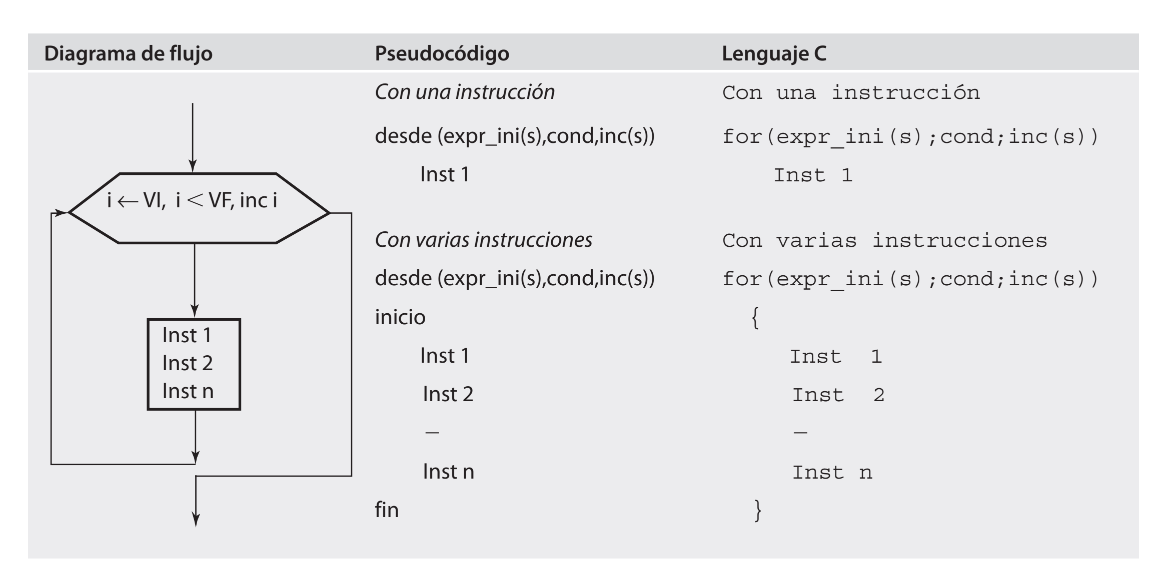


Diagrama de flujo	Pseudocódigo	Lenguaje C
	Con una instrucción	`Con una instrucción`
	desde (expr_ini(s),cond,inc(s))	`for(expr_ini(s);cond;inc(s))`
	Inst 1	`Inst 1`
	Con varias instrucciones	`Con varias instrucciones`
	desde (expr_ini(s),cond,inc(s))	`for(expr_ini(s);cond;inc(s))`
	inicio	`{`
	Inst 1	`Inst 1`
	Inst 2	`Inst 2`
	–	`–`
	Inst n	`Inst n`
	fin	`}`

donde:

expr_ini(s): expresión(es) de asignación que se utilizan para iniciar la(s) variable(s) de control del bucle.
cond: es una expresión relacional o lógica (booleana) que determina cuándo finalizará el ciclo o bucle. La condición puede ser simple o compuesta (una o varias).
inc(s): define cómo cambiará(n) la(s) variable(s) de control cada vez que se repite el bucle o ciclo.

Las tres componentes pueden tener una o varias instrucciones, las cuales deben ir separadas por comas. La *cond* nos lleva al valor final.

Omisión de expresiones

Cualquiera de los componentes en los paréntesis se puede omitir, incluso los tres, pero los separadores *coma (punto y coma)* deben aparecer siempre.

Las tres expresiones del bucle *desde (for)* se pueden omitir, con el siguiente resultado:

Se omite	**Resultado**
Expresión_inicial	No se hace nada antes del bucle
Condición	La condición es siempre cierta (1)
Incremento o expresión_de_paso	No se hace nada tras cada iteración

Ejemplos



Pseudocódigo	**Lenguaje C**
desde (, resultado<>−1 ,)	`for ( ; resultado!=−1 ; )`
inicio	`{`
…	`...`
fin	`}`
desde (, ,)	`for ( ; ; )`
inicio	`{`
/* Bucle infinito */	`/* Bucle infinito */`
fin	`}`

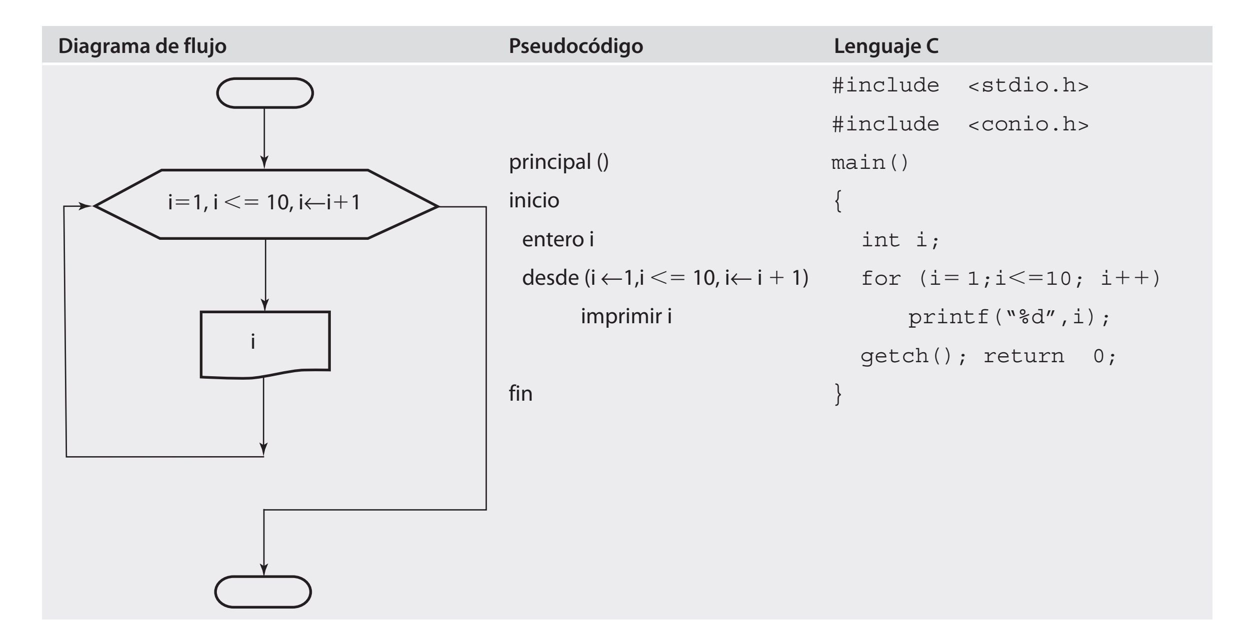
En la instrucción *desde (for)*, primero se ejecutará la(s) inicialización(es), posteriormente se evaluará la condición y, en caso de ser verdadera (no cero), se ejecutará(n) la(s) instrucción(es) que compone(n) el ciclo. Después, se realizará el incremento(s) y se volverá a verificar la condición. Cuando la condición se vuelve falsa, en la siguiente evaluación se terminará el *desde (for)*. Si la condición nunca se vuelve cero, la estructura nunca terminará y el ciclo se repetirá indefinidamente hasta que se detenga en forma manual.

Características:

1. Se debe conocer por anticipado el valor de la variable inicial y final antes de entrar al ciclo.
2. La condición se evalúa antes del bloque de instrucciones. Si la condición es verdadera se ejecuta el bloque, y si es falsa se sale y pasa el control a la instrucción siguiente al bloque.
3. No se debe cambiar el valor de la(s) variable(s) de control, del valor inicial ni del valor final dentro del ciclo.
4. Se puede incrementar o decrementar la variable de control según se requiera.
5. El incremento o decremento de la variable de control es automático.
6. Sólo si existe más de una instrucción dentro del ciclo *desde* se necesita el inicio-fin ({-}).
7. Puede tener una o varias expresiones de inicialización, de condición y de incremento; estas expresiones se separan mediante comas en pseudocódigo y con puntos y comas en lenguaje C.
8. Puede tener alguna o todas las expresiones vacías: desde (,,)-for (; ;).
9. Si la condición está vacía, tenemos un bucle infinito.

3.6.7 Ejercicios resueltos de la estructura de control repetitiva desde (for)

Ejercicio 1. Imprimir en pantalla los primeros 10 números enteros positivos.



Pseudocódigo

```
principal ()
inicio
  entero i
  desde (i ←1,i <= 10, i← i + 1)
        imprimir i
fin
```

Lenguaje C

```c
#include  <stdio.h>
#include  <conio.h>
main()
{
   int i;
   for (i= 1;i<=10; i++)
       printf("%d",i);
   getch(); return  0;
}
```

En el ejercicio 1 antes de entrar al ciclo *desde*, se declara la variable *i*, ésta inicia con el valor de 1, se evalúa la condición *(i<=10)*, es decir *(1<=10)*, como es verdadera, entra al ciclo e imprime el 1, en seguida la *i* se incrementa en 1 *(i←i+1)* convirtiéndose en 2; en la segunda vuelta se evalúa la condición *(2<=10)*, como es verdadera entra al ciclo e imprime el 2, en seguida la *i* se incrementa en 1 y se convierte en 3; de la tercera a la décima vuelta todas las condiciones son verdaderas *(3<=10, 4<=10, 5<=10, 6<=10, 7<=10, 8<=10, 9<=10, 10<=10)* y se imprime 3 4 5 6 7 8 9 y 10; en la décima vuelta después de imprimir el 10, cuando la *i* se incrementa en 1 y vale 11, la condición *(11<=10)* será falsa y por lo tanto se rompe el ciclo (deja de dar vueltas).

```
principal ()
inicio
      entero i
      desde (i← 1,i < 11,i ←i + 1)
             imprimir i
fin
```

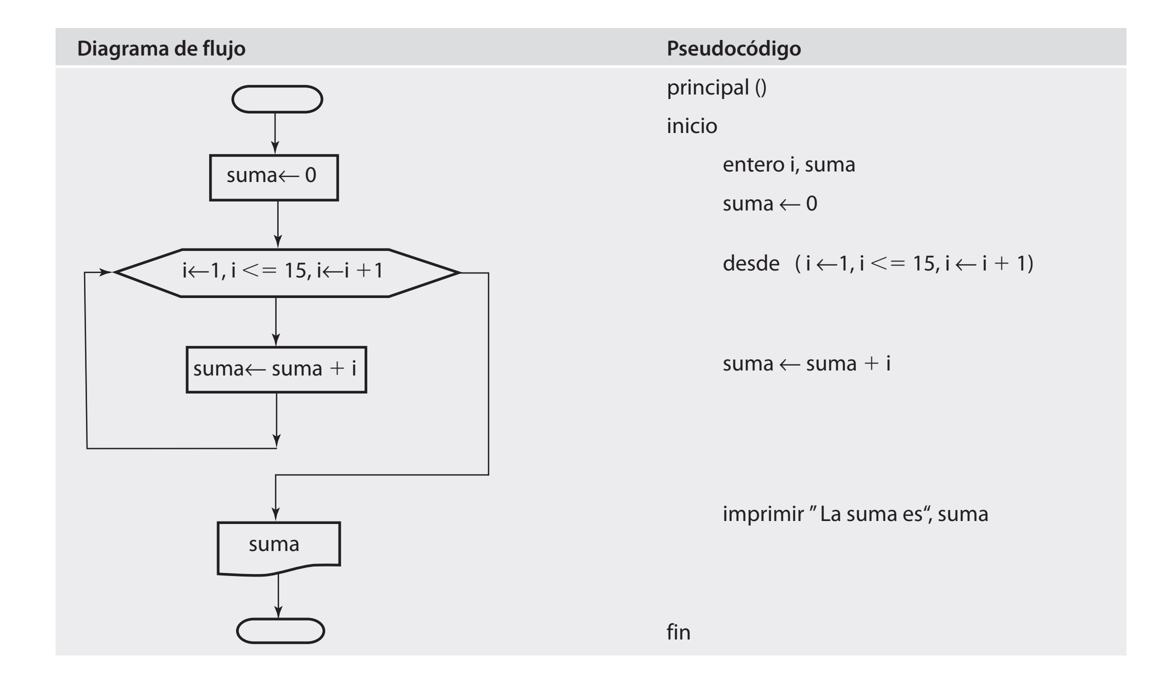
El programa anterior realiza exactamente lo mismo, la diferencia es que las condiciones que revisa son: *(1<11), (2<11), (3<11), (4<11), (5<11), (6<11), (7<11), (8<11), (9<11), (10<11)*, todas ellas son verdaderas y se imprime: 1 2 3 4 5 6 7 8 9 y10; en la décima vuelta después de imprimir el 10, cuando la *i* se incrementa en 1 y vale 11, la condición (11<11) será falsa y por lo tanto se rompe el ciclo (deja de dar vueltas).

En este programa la *variable contador* del ciclo es *i*. Las tres partes del for son:

La *expresión de inicialización* es ***i=1***, la *condición* es (***i<=10***), dicha condición nos lleva al valor final 10, y el *incremento* es ***i++***, es decir que a la *i* en cada vuelta se le suma 1.

Nota: La instrucción for no debe llevar punto y coma al final del paréntesis: for (i=1;i<=10;i++) ya que no respetará la impresión de i dentro del ciclo, imprimirá el valor de *i* que rompió el ciclo, en este caso 11.

Ejercicio 2. Calcule la sumatoria de todos los números entre 1 y el 15, e imprima el resultado.



En el ejercicio 2, antes de entrar al ciclo *desde* se declaran las variables que se van a utilizar, se le indicará al ciclo cuántas veces pasar o entrar (15 veces) y se realizará la instrucción *suma = suma + i*. Ésta es la única instrucción que se repetirá *15* veces ya que el ciclo *desde* no tiene inicio-fin ({-}), la variable *i* sirve como contador y lleva un control de cuántas veces entra al ciclo *desde*.

Ejemplos



i empieza en 1, es decir *i = 1*, se revisa la condición *(i<=n)*, como 1 es menor que 15, entra al ciclo y se realiza la instrucción suma=suma+i, es decir *suma = 0 + 1*, y queda en la variable *suma = 1*, se incrementa *i* en uno, es decir se queda con el valor de *i = 2*. Se revisa la condición *(i<=15)*, como 2 es menor que 15 entra al ciclo y se realiza la instrucción suma = suma + i, es decir *suma =1+2*, queda en la variable *suma = 3*, se incrementa *i* en uno, es decir se queda con el valor de *i=3*, así sucesivamente hasta que *i =15*. Se revisa la condición *(i<=15)*, como 15 es menor o igual a 15, entra al ciclo y se realiza la instrucción suma = suma + i, es decir *suma=105 +15*, queda en la variable *suma = 120*, se incrementa *i* en uno, es decir se queda con el valor de *i = 16*. Se revisa la condición *(i<=16)*, como 16 no es menor ni igual a 15, se sale imprimiendo del ciclo *desde* e imprime *La suma es 120*.

Ejercicio 2

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i, suma;
    suma = 0;
    for(i=1;i<=15;i++)
       suma = suma+i;
    printf("La suma es %d",suma);
    getch();
    return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

La suma es 120

En este programa la *i* es una variable *contador* ya que en cada vuelta se incrementa de uno en uno (de manera constante) y la suma es un *acumulador* ya que en cada ciclo se incrementa en cantidades variables (los valores de *i* van cambiando).

Ejercicio 2a

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i, suma =0;
    for(i= 1;i<=15;i++)
    {
        suma +=i;
        printf("La suma es %d\n",suma);
    }
    getch();
    return  0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

La suma es 1
La suma es 3
La suma es 6
La suma es 10
La suma es 15
La suma es 21
La suma es 28
La suma es 36
La suma es 45
La suma es 55
La suma es 66
La suma es 78
La suma es 91
La suma es 105
La suma es 120

Existen programas donde nos va a interesar conocer las acumulaciones parciales; a diferencia del ejercicio 2, en el 2a se imprimen los valores de la sumatoria parcialmente, ya que la impresión está dentro del ciclo.

Ejercicio 3. Calcule la sumatoria de todos los números entre 1 y un número leído por el usuario (n) e imprima el resultado.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero i, suma, n	`int i, suma =0, n;`
suma ← 0	`printf("Dame un número entero ");`
imprimir "Dame un número entero"	`scanf("%d",&n);`
leer n	`for(i=1;i<=n;i++ )`
desde (i ←1, i <= n, i ← i + 1)	`suma +=i;`
suma ← suma + i	`printf("La suma es %d",suma);`
imprimir" La suma es ", suma	`getch(); return 0;`
fin	`}`

Una posible salida en pantalla, después de ejecutar el programa sería: Dame un número entero 9 La suma es 45

Este ejercicio 3 es similar al ejercicio 2, la diferencia es que antes de entrar al ciclo *desde* se lee un número y se guarda en la variable *n*, esto le indicará al ciclo cuántas veces pasar o entrar y realizar la instrucción suma = suma + i. Ésta es la única instrucción que se repetirá *n* veces ya que el ciclo *desde* no tiene inicio-fin ({-}), la variable *i* sirve como contador y para llevar un control de cuántas veces entra al ciclo *desde*.

Ejemplos



Si *n = 6*, i empieza en 1, es decir *i = 1*, se revisa la condición *(i<=n)*, como 1 es menor que 6, entra al ciclo y se realiza la instrucción suma = suma + i, es decir *suma = 0 + 1*, y queda en la variable *suma = 1*, se incrementa *i* en uno, es decir se queda con el valor de *i = 2*. Se revisa la condición *(i<=n)*, como 2 es menor que 6, entra al ciclo y se realiza la instrucción suma = suma + i, es decir *suma = 1 + 2*, quedando en la variable *suma = 3*; se incrementa *i* en uno, es decir se queda con el valor de *i = 3* y así sucesivamente hasta que *i = 6*. Se revisa la condición *(i<=n)*, como 6 es menor o igual a 6, entra al ciclo y se realiza la instrucción suma = suma + i, es decir *suma = 15 + 6* y queda en la variable *suma = 21*; se incrementa *i* en uno, es decir se queda con el valor de *i = 7*; se revisa la condición *(i<=n)*, como 7 no es menor ni igual a 6, se sale imprimiendo *La suma es 21*.

Ejercicio 4. Imprimir todas las letras del alfabeto en mayúsculas.

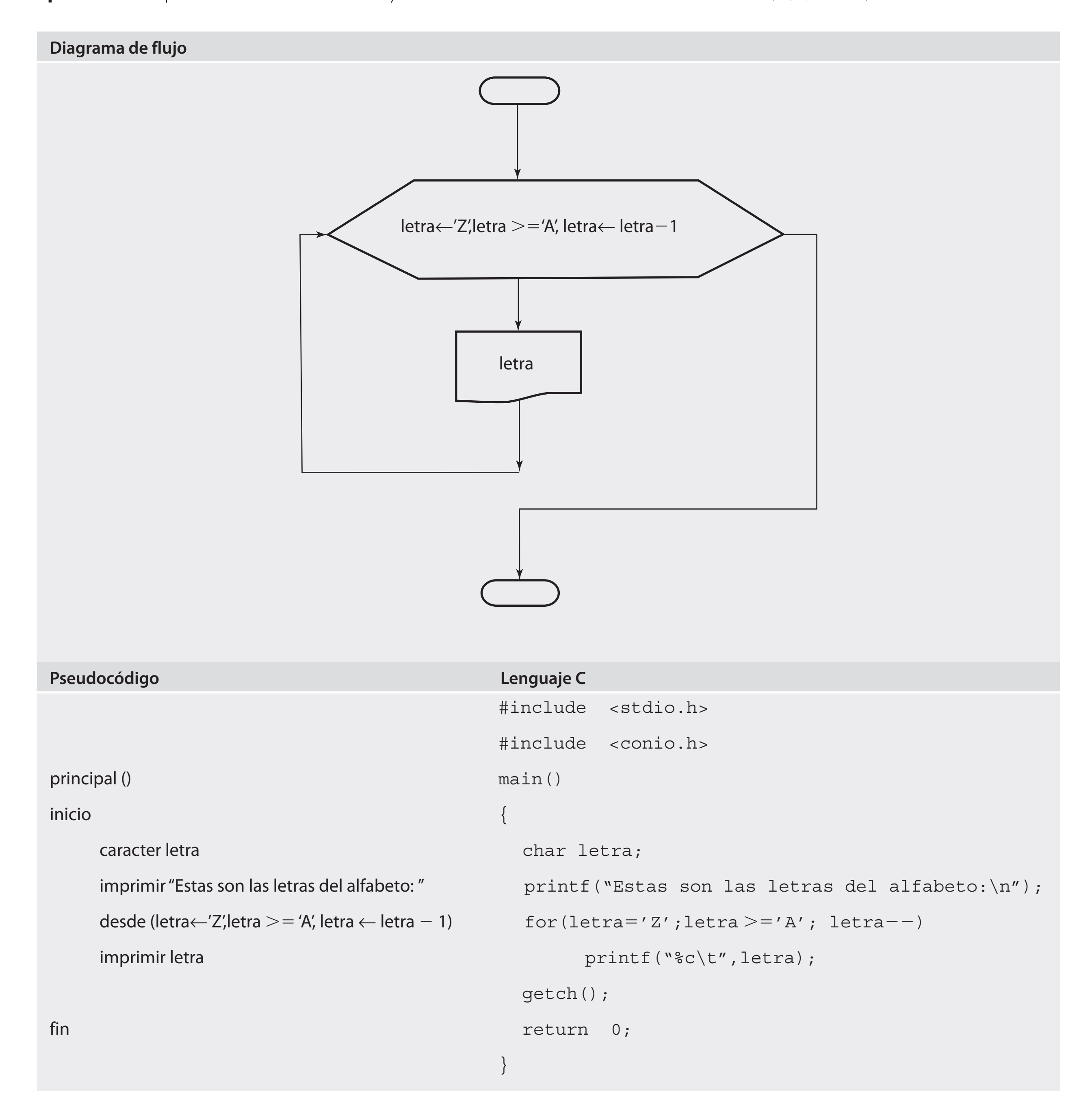
Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
caracter letra	`char letra;`
imprimir "Estas son las letras del alfabeto: "	`printf("Estas son las letras del alfabeto:\n");`
desde (letra←'A',letra <= 'Z', letra ← letra + 1)	`for(letra='A';letra<='Z';letra ++)`
imprimir letra	`printf("%c\t",letra);`
	`getch();`
fin	`return 0;`
	`}`

En el ejercicio 4 la variable *contador*, en este caso letra, también puede ser un carácter; en este ejemplo en la primera vuelta la variable letra inicia con el valor de A, se evalúa la condición *(letra<='Z)*. Observe que el valor del carácter letra se escribe entre apóstrofos: 'A', es decir *('A' < 'Z')*, como es verdadera entra al ciclo e imprime la A, en seguida la letra se incrementa en 1 (letra←letra+1) y se convierte en B. En la segunda vuelta se evalúa la condición *('B'<='Z')*, como es verdadera entra al ciclo e imprime la B, en seguida la letra se incrementa en 1 (letra←letra+1) y se convierte en Z; continúa con el mismo proceso, e imprime: *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z*, en la última vuelta después de imprimir Z, cuando la letra se incrementa en 1 y vale `[`, aunque se nos haga raro el siguiente carácter es `[`, debido a que seguimos la secuencia del código ASCII; la condición *(`[`<=`Z`)* será falsa ya que el equivalente sería *(91<=90)* y por lo tanto se rompe el ciclo. El código ASCII de la `A` es 65, de la `Z` es 90, el siguiente es el carácter especial `[` del cual su código ASCII es el 91.

Nota: No aparece la Ñ debido a que el código ASCII es 165 y el alfabeto está comprendido entre los códigos 65 y 90, por lo que queda fuera de ese rango.

En lenguaje C es una secuencia de escape que mueve el cursor a la posición siguiente del tabulador horizontal (vea anexo A).

Ejercicio 5. Imprimir todas las letras mayúsculas del alfabeto de manera invertida (Z,Y,X....A).



Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
caracter letra	`char letra;`
imprimir “Estas son las letras del alfabeto: ”	`printf(“Estas son las letras del alfabeto:\n”);`
desde (letra←’Z’,letra >= ‘A’, letra ← letra − 1)	`for(letra=’Z’;letra >=’A’; letra−−)`
imprimir letra	`printf(“%c\t”,letra);`
	`getch();`
fin	`return 0;`
	`}`

En este ejercicio, al igual que en el anterior la variable *contador* es un carácter; en este ejemplo en la primera vuelta la variable letra inicia con el valor de Z, se evalúa la condición *(letra>=‘A’)*, es decir ‘Z’ > ‘A’, como es verdadera entra al ciclo e imprime la Z, en seguida la variable letra se decrementa en 1 (letra←letra − 1) y se convierte en Y; en la segunda vuelta: se evalúa la condición *(Y >= ’A’)*, como es verdadera entra al ciclo e imprime la Y, en seguida la letra se decrementa en 1 (letra←letra − 1) y se convierte en Z; continúa con el mismo proceso e imprime: *Z Y X W V U T S R Q P O N M L K J I H G F E D C B A*; en la última vuelta después de imprimir A, cuando la letra se decrementa en 1 y vale @, aunque se nos haga raro el siguiente carácter es @, debido a que seguimos la secuencia del código ASCII, la condición *(’@’ >= `A`)* será falsa ya que *(64>=65)* y por lo tanto se rompe el ciclo.

Ejercicio 6. Imprimir en pantalla la tabla de multiplicar de cualquier número entero (n).

Pseudocódigo	Ejercicio 6a Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero i, n	`int i, n;`
imprimir "Dame el número de tabla :"	`printf("Dame el número de tabla:");`
leer n	`scanf("%d",&n);`
desde (i ←1,i<=10,i ←i + 1)	`for(i=1;i<=10;i++)`
imprimir n * i	`printf("%d*%d=%d\n", i,n,n*i);`
	`getch(); return 0;`
fin	`}`

En este ejercicio primero se pregunta el número de tabla que se desea imprimir y se guarda en la variable *n*, entra al ciclo *desde*, en el cual por cada valor que toma *i* lo usa y lo multiplica por el número que capturó el usuario y se almacenó en *n*, nótese que el resultado no se guarda en una variable, sólo se manda a imprimir *n * i*.

Ejemplos



Si *n = 5* entra al ciclo *desde* y la variable *i* se inicializa en 1, como se cumple la condición de *(1 <=10)*, entra al ciclo e imprime *n * i*, es decir 5*1, sólo que primero realiza la multiplicación y muestra solamente el valor 5, incrementa *i* en 1 y queda 2. Como se cumple la condición de *(2 <=10)*, entra al ciclo e imprime *n * i*, es decir 5*2, el valor impreso será el 10 y así sucesivamente hasta que la variable *i* tome el valor de 10; es la última vez que se cumple la condición de *(10 <=10)*, entra al ciclo e imprime *n * i*, es decir 5*10, el último valor que imprima será el 50. Así se imprimirán en pantalla los valores *5 10 15 20 25 30 35 40 45 50*.

La diferencia entre el ejercicio 6 y el 6a es que en lenguaje C no sólo imprimimos el resultado de la expresión *n*i*, sino toda la tabla con: printf("%d*%d=%d\n",i,n,n*i);

Ejercicio 7. Elevar un número X a una potencia Y e imprimir el resultado.

Pseudocódigo
principal ()
inicio
entero cont,x,y,res←1
imprimir "Escriba el número que se elevará a una potencia: "; leer x
imprimir "¿A qué potencia elevará el número?"; leer y
desde (cont←1, cont<=y, cont←cont+1)
res←res * x
imprimir "El ",x," elevado a la potencia ",y," es: ",res
fin

En este ejercicio se declaran las variables *x*, que equivale a la base, y la potencia a la cual se desea elevar la base, *cont* para llevar el control del ciclo *desde* y *res*, que estará almacenando los valores parciales en cada vuelta del ciclo y cuando termine y se salga del ciclo el valor final de la operación, es decir la potencia *res* se

inicializa con 1 ya que se realiza en cada vuelta del ciclo un producto o multiplicación; si no se inicializara en 1 (uno) el valor sería 0 (cero) y la instrucción *res←res* * x sería siempre 0 (cero), *res←0 * 1 = 0*, por esta razón se inicializan siempre las variables que almacenarán productos en 1. Se le pregunta al usuario el valor de la base, se almacena en *x* el valor de la potencia y se almacena en *y*, el *cont* se inicializa en 1, en el ciclo *desde*, teniendo en cuenta que el último valor que tomará el *cont* será igual al de la potencia, para entrar al ciclo. En este programa escribimos dos instrucciones en una sola línea, por lo que las separamos con punto y coma.

Ejemplos



Si *x* = *2* y *y* = *4*, se inicia el ciclo *desde (cont←1, cont <=4, cont←cont +1),* es decir se multiplicará el valor de *x* cuatro veces y cada resultado parcial se estará almacenando en la variable *res*. En la primera vuelta se compara la condición *(cont <=y)*, es decir *(1 <=4)*. Como la condición se cumple entra al ciclo *desde* y se realiza la operación *res←res * x*, sustituyendo valores *res←1 * 2 = 2*, se incrementa el *cont* en 1, así que toma el valor de *cont = 2*, se revisa la condición *(cont <=y)*, es decir *(2 <=4)*. Como la condición se cumple entra al ciclo *desde* y se realiza la operación *res←res * x*, sustituyendo valores *res←2 * 2 = 4,* se incrementa el *cont* en 1, así que toma el valor de *cont = 3*, se revisa la condición *(cont <=y)*, es decir *(3<=4)*. Como la condición se cumple entra al ciclo *desde* y se realiza la operación *res←res * x*, sustituyendo valores *res←4 * 2 = 8*, se incrementa el *cont* en 1, así que toma el valor de *cont = 4*, se revisa la condición *(cont <=y)*, es decir *4 <=4*. Como la condición se cumple entra al ciclo *desde* y se realiza la operación *res←res * x*, sustituyendo valores *res←8 * 2 = 16*, se incrementa el *cont* en 1, así que toma el valor de *cont = 5*, se revisa la condición *(cont <=y)*, es decir *(5 <=4)*. Como la condición no se cumple se sale del ciclo *desde* y se imprime: *El 2 elevado a la potencia 4 es: 16.*

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
    int cont, x, y, res=1;
    printf("Escriba el número que se elevará a una potencia:  ");
    scanf("%d",&x);
    printf("¿A qué potencia elevará el número %d ? ",x); scanf("%d",&y);
    for(cont=1;cont<=y;cont++)
          res=res*x;
    printf("El %d elevado a la potencia %d es: %d\n",x, y, res); getch();
    return 0;
}
```

Ejercicio 8. Calcular el factorial de un número entero e imprimir el resultado.

Pseudocódigo

```
principal ()
inicio
     entero i, x;   enterolargo fact
     fact ←1;
     imprimir " Dame el número para calcular su factorial : ";  leer  x
     desde (i← x, i > 1, i ← i − 1)
          fact←fact * i
     imprimir " El factorial de", x ,"= ", fact
fin
```

El factorial de un número es el producto del mismo número y de todos los valores menores a él hasta llegar a la unidad, es decir si se desea el factorial de 4!= 4 * 3 * 2 *1 teniendo como resultado 24; el último valor, el 1. En este ejercicio las variables *i, x* son de tipo entero pero la variable *fact* deberá ser entero largo ya que el factorial de 8 es 40 320 y no se puede almacenar en un número entero que alcanza hasta el valor 32 637. Primero se pregunta a qué número se le quiere calcular el factorial y se guarda en la variable x, se entra al ciclo *desde* y la variable *i* se inicializa con el mismo valor de *x*, es decir toma el valor del número al que se le quiere calcular el factorial.

Ejemplos



Si *x* = *4*, *i* se inicializa con ese valor, *i* = *4*, se revisa la condición del ciclo, que es *(i >1)*, es decir *(4 >1)*, como se cumple se realiza la instrucción *fact* = *fact *i*, al sustituir los valores *fact* = *1* 4* y queda en la variable *fact* = *4*, se decrementa *i* en 1 y queda *i* = *3*, luego se revisa la condición del ciclo, que es *(i >1)*, es decir *(3 >1)*, como se cumple, se realiza la instrucción *fact* = *fact *i* al sustituir los valores *fact* = *4* 3* y queda en la variable *fact* = *12*, se decrementa *i* en 1 y, queda *i* = *2*. Se revisa la condición del ciclo, que es *(i >1)*, es decir *(2 >1)*, como se cumple, se realiza la instrucción *fact* = *fact *i* sustituyendo los valores *fact* = *12* 2*, y queda en la variable *fact* = *24*, se decrementa *i* en 1, queda *i* = *1*, se revisa la condición del ciclo, que es *(i >1)*, es decir *(1 >1)*, como no se cumple, se sale del ciclo y se imprime la siguiente instrucción que se encuentra después del ciclo *desde*: *El factorial de 4* = *24*.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
   int i, x; long int fact=1;
   printf("Dame el número para calcular su factorial "); scanf("%d",&x);
   for (i=x; i > 1 ; i−−)
   fact=fact * i;
   printf("El factorial de  %d = %ld", x, fact  );
   getch();  return  0;
}
```

La posible salida en pantalla, después de ejecutar el programa sería:

Dame el número para calcular su factorial: 5

El factorial de 5 = 120

Ejercicio 9. En un salón de clases seleccione a los alumnos que se integrarán al equipo de basquetbol; las características para la selección son: para el equipo varonil que su altura sea mayor a 1.75 y para el femenil que su altura sea mayor que 1.65. Imprimir del grupo de clases cuántos fueron los seleccionados.

Pseudocódigo

```
principal ()
inicio
        entero i, alum, eqvar, eqfem
        real alt
        caracter sexo
        eqvar ← eqfem ← 0
        imprimir "¿Dame el número de alumnos del salón de clases ?" leer   alum
        desde (i ← 1, i <= alum, i ← i + 1)
        inicio
                imprimir "Dame la altura y el sexo del alumno:     "   ,i
                leer   alt, sexo
                si ( sexo = 'M' o sexo ='m')
                         si ( alt > 1.75 )   eqvar ← eqvar +1
                si no si ( alt > 1.65 )   eqfem ← eqfem +1
         fin
        imprimir   "El total de alumnos seleccionado para el equipo varonil es" ,eqvar
        imprimir   "El total de alumnos seleccionado para el equipo femenil es" ,eqfem
fin
```

En este ejercicio las variables *eqvar* y *eqfem* se inicializan con 0, ya que servirán como contadores. Se utiliza el ciclo repetitivo *desde* para hacer la encuesta entre los alumnos de una clase, preguntando primero el total de alumnos y almacenándolo en la variable alum; entra al ciclo *desde* y a cada alumno se le pregunta su altura y su sexo guardándolo respectivamente en las variables alt y sexo; la variable alt es real ya que la mayoría de los alumnos miden entre 1.50 y 1.95; y la variable sexo es de tipo caracter, ya que almacenaremos "M" si el alumno es masculino y "F" si es femenino. Primero revisamos el sexo; si es masculino entra a la parte verdadera de la estructura selectiva y evalúa la otra condición, para verificar la altura *(alt>1.75)*. Si esta condición se cumple se incrementa el contador del equipo varonil en uno, eqvar ← eqvar +1; si el sexo no es masculino "M" se entenderá que es femenino "F" y se pasará a la parte falsa de la primera condición, es decir al si no y se evaluará la condición *(alt >1.65)*, si esta condición se cumple se incrementa el contador del equipo femenil en uno, eqfem ← eqfem+1; todo esto se realizará por cada uno de los alumnos ya que al terminar de revisar la condición se incrementa *i* en uno *i=2* y se regresa al ciclo *desde* y le pide los datos al segundo alumno, así sucesivamente hasta terminar con el último alumno.

Ejemplos



"Dame el número de alumnos del salón de clases?", 5, la variable *alum= 5*. Al primer alumno se le pregunta "Dame la altura y el sexo del alumno", *alt 1.80* y *sexo= "M",* se cumple la primera condición del sexo y también la condición de la altura, así que el *eqvar=0+1=1*; el segundo alumno captura *alt 1.60* y *sexo = "F"*, como no se cumple la primera condición del sexo pasa al si no, y no se cumple la condición de la altura así que el *eqfem=0*; el tercer alumno captura *alt 1.68 y sexo= "F"*, no se cumple la primera condición del sexo así que pasa al si no y se cumple la condición de la altura, así que el *eqfem=0+1=1*; el cuarto alumno captura *alt 1.67 y sexo= "M"*, se cumple la primera condición del sexo pero no la condición de la altura así que el *eqvar* se queda igual con *eqvar=1*; el quinto alumno captura *alt 1.72 y (sexo = "F")*, no se cumple la primera condición del sexo así que pasa al si no pero se cumple la condición de la altura, así que el *eqfem=1+1=2*, si vemos la variable *i* = 6, cuando revisa la condición del *desde* no se cumple y se sale del ciclo e imprime El total de alumnos seleccionado para el equipo varonil es 1. *El total de alumnos seleccionado para el equipo femenil es 2.*

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
   int i, alum, eqvar, eqfem;
   float alt;
   char sexo;
   eqvar = eqfem = 0;
   printf("¿Dame el número de alumnos del salón de clases?");
   scanf("%d",&alum);
   for (i = 1; i <= alum; i++)
   {
      printf("Dame la altura del alumno %d: ",i);
      scanf("%f",&alt);
      printf("Dame su sexo : ");
      fflush(stdin); sexo=getchar();
      if  ( sexo=='M'|| sexo=='m')
         { if (alt > 1.75) eqvar = eqvar +1; }
      else if (alt > 1.65 ) eqfem = eqfem +1;
   }
   printf("El total de alumnos seleccionado para el equipo varonil es: %d\n",eqvar);
   printf("El total de alumnos seleccionado para el equipo femenil es: %d " ,eqfem);
   getch(); return  0;
}
```

En lenguaje C se pueden inicializar varias variables con el mismo valor: eqvar = eqfem = 0, se le llama sobrecarga de operadores. Como hemos visto, el lenguaje C en ocasiones tiene problemas con el formato %c para las cadenas, por lo que hubo necesidad de utilizar las funciones fflush (stdin) y getchar(). Las llaves son necesarias para indicar que el else es de la condición del sexo.

3.6.8 Casos especiales del for

Esta sección se verá sólo en lenguaje C, ya que los casos especiales son propios de dicho lenguaje.

Recuerde la sintaxis: for (expr_ini(s); cond ;inc(s))

Como se comentó anteriormente en el *for* pueden faltar alguna o todas las expresiones, pero deben permanecer los signos de punto y coma. Los siguientes dos ejemplos son equivalentes:

Ejemplos



Ejemplo 1	Ejemplo 2
`#include <conio.h>`	`#include <conio.h>`
`#include <stdio.h>`	`#include <stdio.h>`
`main()`	`main()`
`{`	`{`
`int i=1, suma=0;`	`int i=1, suma=0;`
`for (; i<=10; ++i) suma += i;`	`for (; i<=10 ;) suma += i++;`
`printf("%d\n", suma);`	`printf("%d\n", suma);`
`getch(); return 0;`	`getch(); return 0;`
`}`	`}`

Recuerde que i++ equivale a decir: úsala y luego increméntala (vea el capítulo 2).

Cuando falta la expresión *cond*, se le supone valor verdadero siempre y se forma un ciclo infinito, como lo apreciamos en el ejemplo 3*a*); en el ejemplo 3*b*) vemos la versión corregida:

Ejemplos



Ejemplo 3*a*)

```
#include <conio.h>
#include <stdio.h>
main()
{
    int i=1, suma=0;
    for ( ; ; )
    { suma += i++;
      printf("%d\n", suma);
    }
    getch(); return 0;
}
```

Ejemplo 3*b*)

```
#include <conio.h>
#include <stdio.h>
main()
{
    int i=1, suma=0;
    for ( ;i<=10  ;  )
    {  suma += i++;
       printf("%d\n", suma);
    }
    getch(); return 0;
}
```

for (i = 1; i <= 100; i ++); //provoca un retardo de 100 ciclos.

El operador coma " , "

Es un operador binario para expresiones : *expresion1, expresion2, ...* y se utiliza tanto en **expr_ini**(s), **cond**, como en **inc**(s):

Se evalúa primero "expresion1", después "expresion2" y así sucesivamente.

Ejemplos



en **expr_ini**(s) podemos tener: suma=0 , i=1

Se suele usar en las sentencias " for " porque permite hacer asignación múltiple de valores iniciales y procesamiento múltiple de incrementos.

Los ejemplos 1 y 2 los podemos escribir de la siguiente forma:

Ejemplos



```
       Ejemplo 1a)                              Ejemplo 2a)
for (suma=0, i=1; i<=10; ++i)      for (suma=0,i=1;i<=10; suma += i, ++i)
suma += i;                         printf("%d\n", suma);
printf("%d\n", suma);
```

3.6.9 Ejercicios resueltos de la estructura de control repetitiva en casos especiales desde (for)

Ejercicio 1. Imprimir la suma de los primeros 3 enteros pares y la de los primeros 3 enteros impares:

```
#include <stdio.h>
#include <conio.h>
main()
{  int par=0, impar=0;
   for (int c=1, p=2, i=1; c<=3; c++, p+=2, i+=2)
   {
      par+=p;
      impar+=i;
   }
   printf("%d, %d\n",par,impar);
   getch(); return  0;
}
```

En el ejercicio 1 inicializamos los acumuladores *par* e *impar* con cero, ya que acumularemos sumandos. Declaramos e inicializamos las variables *c*, *p* e *i* dentro del for. La variable c es un contador que comienza en 1 y termina en 3 (total de pares e impares que se imprimirán). La variable *p* representa los valores pares comenzando con 2 y la variable *i* representa los valores impares comenzando con 1. Como la *c* inicia con 1 la condición *(1<=3)* es verdadera entra al ciclo, calculando los valores de *par=0+2=2* e *impar=0+1=1*. Se incrementa en 1 la variable *c*, como la condición *(2<=3)* es verdadera entra al ciclo, calculando los valores de *par=2+4=6* e *impar=1+3=4*. Se incrementa de nuevo en 1 la variable *c*, la condición *(3<=3)* es verdadera entra al ciclo, calculando los valores de *par=6+6=12* e *impar=4+5=9*. Se incrementa de nuevo en 1 la variable *c*, la condición *(4<=3)* es falsa por lo que se rompe el ciclo y se imprimirá 12, 9.

Ejercicio 2. Imprimir el alfabeto alternando mayúsculas y minúsculas: Aa, Bb, Cc, Dd, Ee, Ff, Gg, Hh, Ii, Jj, Kk, Ll, Mm, Nn, Oo, Pp, Qq, Rr, Ss, Tt, Uu, Vv, Ww, Xx, Yy, Zz.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
   char caracter,c;
   for (caracter='a',c='A';caracter<='z';caracter++,c++)
   {
       printf("%c",c);
       printf("%c,",caracter);
   }
   getch(); return 0;
}
```

En este ejercicio inicializamos la variable *caracter* con la *a* minúscula y la variable *c* con la *A* mayúscula. Como la letra *a* es menor que la letra *z* en el código ASCII, la condición es verdadera por lo que entra al ciclo imprimiendo *Aa*. Se incrementan en uno las variables *caracter* y *c*, tomando los valores de *b* y *B*, respectivamente; como la letra *b* es menor que la letra *z* en el código ASCII, la condición es verdadera por lo que entra al ciclo imprimiendo *Bb*. Se repetirá el proceso con todas las letras hasta que llegue a la letra *z* minúscula imprimiendo *Zz*. Cuando se incrementan de nuevo la variable *caracter* toma el valor de "{" (siguiente carácter en el código después de la z minúscula). Dicho carácter es mayor a la *z*, por lo que se rompe el ciclo y el programa termina.

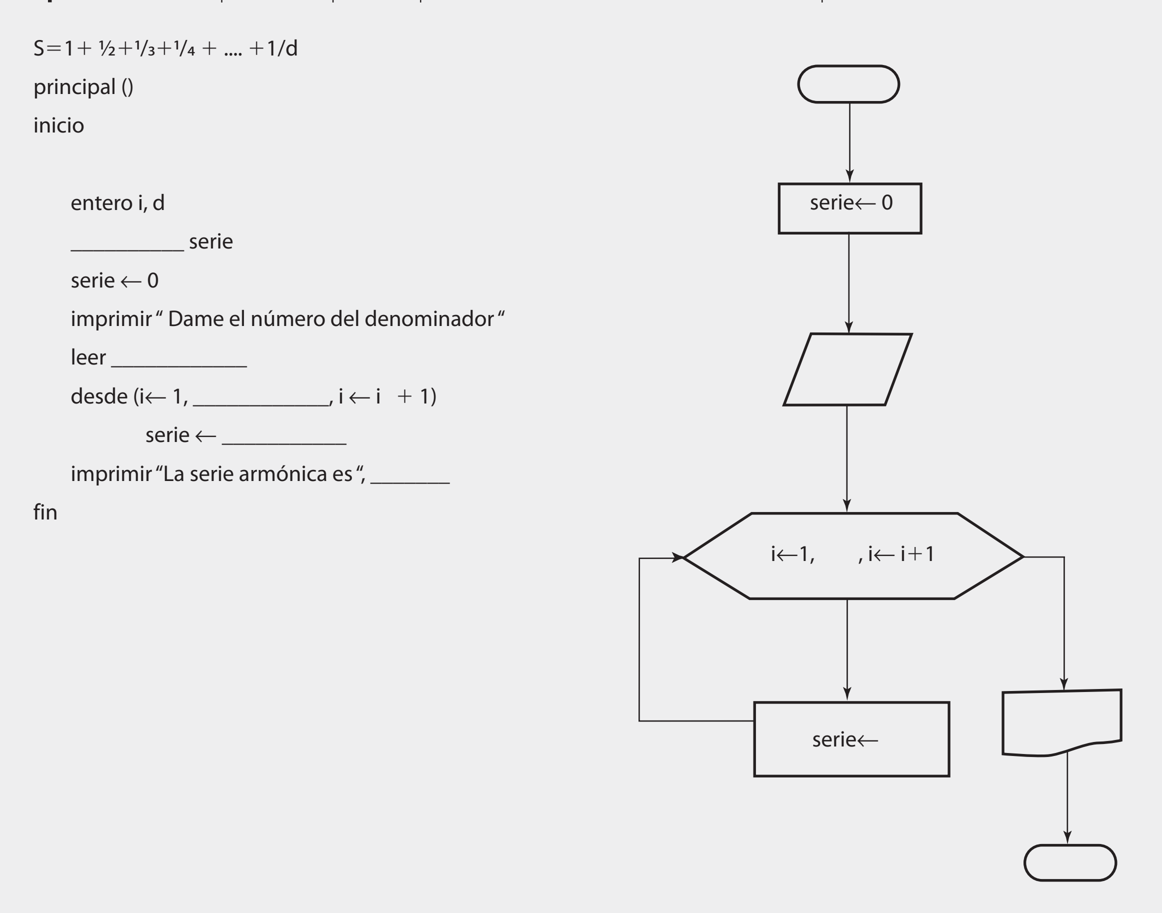
Ejercicios complementarios de la estructura de control repetitiva desde en pseudocódigo

Ejercicio 1. Completar el ejercicio para calcular la serie armónica e imprimir el resultado.

$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + 1/d$

```
principal ()
inicio

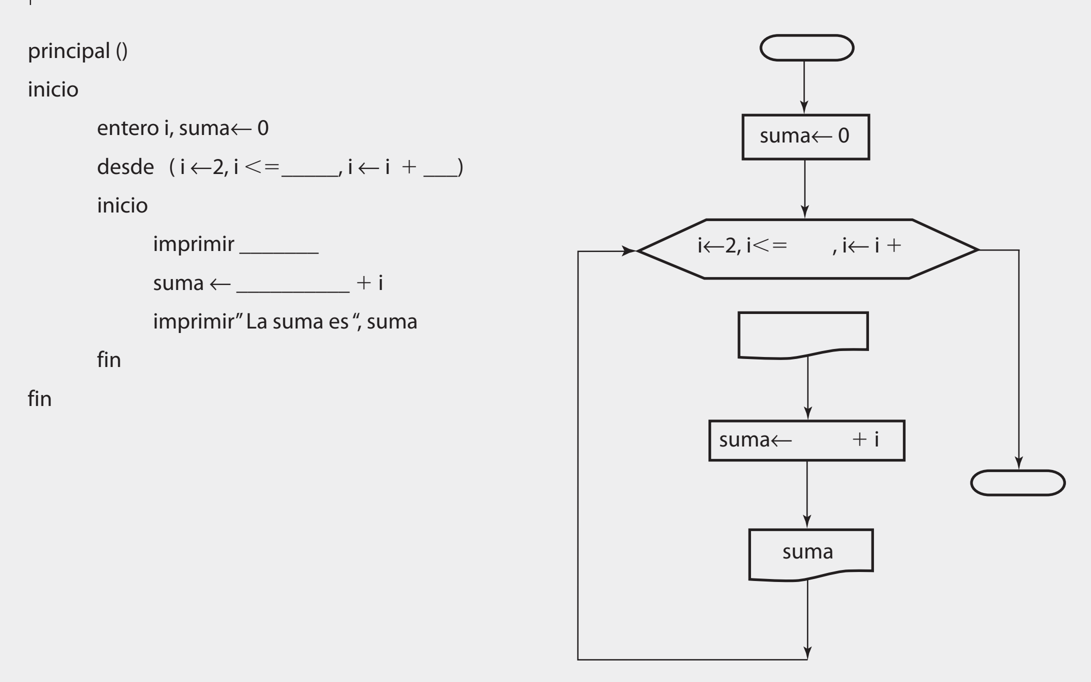
    entero i, d
    ___________ serie
    serie ← 0
    imprimir " Dame el número del denominador "
    leer _____________
    desde (i← 1, _____________, i ← i   + 1)
            serie ← ____________
    imprimir "La serie armónica es ", ________
fin
```



Si d = 5 con qué valores se quedan las variables i y serie, i = ___________ serie = ___________

Ejercicio 2. Complete el siguiente pseudocódigo y el diagrama de flujo para que se impriman en pantalla los números pares entre 1 y 300 y los vaya acumulando (sumando) en una variable. Que se imprima en pantalla el resultado de la suma.

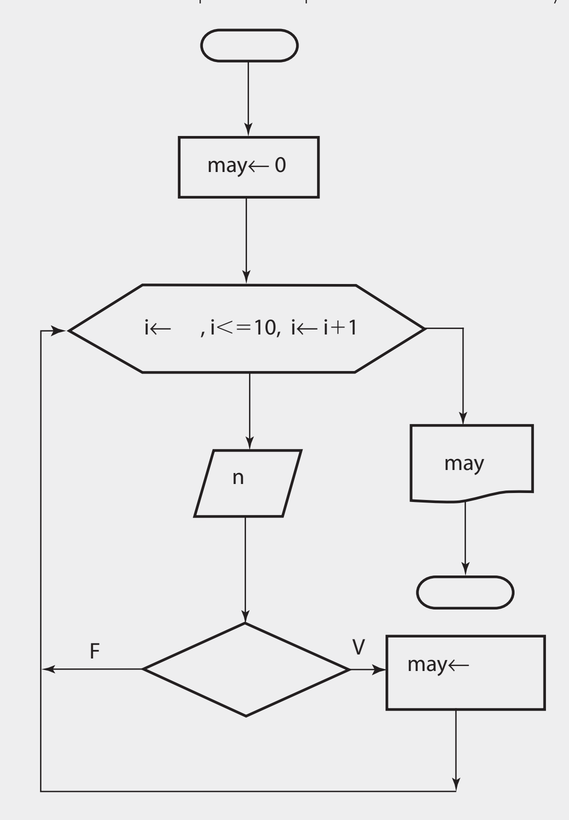
```
principal ()
inicio
        entero i, suma← 0
        desde  ( i ←2, i <=_____, i ← i  + ___)
        inicio
              imprimir _______
              suma ← __________ + i
              imprimir" La suma es ", suma
        fin
fin
```



Ejercicio 3. Complete el siguiente pseudocódigo y el diagrama de flujo para encontrar el número mayor entre un conjunto de 10 números enteros positivos. Que se imprima en pantalla el número mayor.

```
principal ()
inicio
      entero i, _____, may ← 0

      desde (i←_____, i  <= 10, i ← i + 1)
      inicio
            imprimir " Dame el número?  "
            leer  n
            si (_______________)
                  may ← ______
      fin
      imprimir " El número mayor es ", may
fin
```



Ejercicio 4. Complete el siguiente pseudocódigo y realice el diagrama de flujo para calcular e imprimir las 10 primeras potencias de un número *n*.

```
principal ()
inicio
    entero cont,_______, ________
    pot←1
    imprimir "Escriba el número que se elevará a una potencia:"
    leer x
    desde (cont←1, cont <=_______, cont←cont +1)
    inicio
        pot←__________ * x
        imprimir "El ",x," elevado a la potencia" ,________," es: ",_________
    fin
fin
```

Ejercicio 5. Complete el siguiente pseudocódigo y realice el diagrama de flujo para calcular el promedio entre un conjunto de m materias.

```
principal ()
inicio
        entero m, ________
        real cal, sum, __________
        sum ← 0
        imprimir "Dame el número de materias cursadas"
        leer   m
        desde (___________________________________)
        inicio
            imprimir "Dame la calificación de la materia"   , i
            leer
            sum← sum + ______
        fin
        prom ← ___________________
        imprimir " El promedio es " ,____________
fin
```

Ejercicios complementarios de la estructura de control repetitiva *for* en lenguaje C

Parte I. Detectar y marcar los errores del siguiente programa, el cual suma los 100 primeros números enteros positivos

```
float main()
{
   int contador,suma=1;
   for(contador==1;contador< =100;Contador+-);
       suma =suma+contador;
   printf("La suma es : %d",suma);
   getch();
}
```

Parte II. Completar el siguiente programa, el cual imprime las 10 primeras potencias de un número x

```
main()
{
   float base,___________________;
   int contador;
   printf("Introduzca el número para imprimir sus 10 primeras potencias\n");
   scanf("%f",&base);
   for  (contador=1;__________________;contador++)
   {
       ____________________________________________
       printf("El resultado es:  %.2f\n",resultado);
   }
   getch();    return  0;
}
```

Parte III. Elaborar la codificación en lenguaje C de los siguientes programas utilizando la instrucción for (realizando el cálculo e impresión respectiva):

1. Leer *n* números e imprimir cuántos son positivos, cuántos negativos y cuántos fueron cero.
2. Imprimir los números del uno al cien excepto los múltiplos de 7, es decir: 1 2 3 4 5 6 8 9 10 11 12 13 15 16 17 18 19 20 22…
3. Leer *n* números enteros y calcular la suma, promedio, producto, el mayor y el menor.
4. Leer *n* números e imprimir el mayor de los múltiplos de 5 leídos y el menor de los múltiplos de 3 leídos.
5. Calcular el promedio de los números múltiplos de 9 que hay de 45 a 194.
6. Imprimir la tabla del código ASCII, valor decimal, carácter, hexadecimal y octal (vea el anexo D).
7. Imprimir un marco de asteriscos alrededor de la pantalla, dichos asteriscos aparecerán de diferente color, de manera lenta y en el sentido de las manecillas del reloj.

8. Leer el número (n) de asteriscos de una línea e imprimir en el centro de la pantalla una línea horizontal.
9. Leer 2 coordenadas (validarlas), la primera es la esquina superior izquierda y la segunda la inferior derecha. Dibujar un marco de asteriscos con los datos proporcionados.
10. Generar los primeros treinta números de la serie de Fibonacci.
11. Generar aleatoriamente 5 operaciones aritméticas consecutivas; los números (aleatorios) sobre los que se realizan las operaciones deben estar comprendidos entre 1 y 10 y las operaciones aritméticas (también aleatorias) permitidas serán suma, resta, multiplicación y división (en el caso de la división se realizará siempre una división entera y el primer número tendrá que ser mayor al segundo). Al final imprimir el número de aciertos y la calificación correspondiente.

Ejemplos



5 × 9 = 45

16 + 4 = 23

6 * 3 = 18

36 − 7 = 4

6 / 3 = 2

Tuviste 3 aciertos, tu calificación es 60.

12. Calcular los cuadrados, los cubos y las raíces cuadradas de los números del 1 al 9 y que utilice tabuladores para imprimir la siguiente tabla de valores:

Número	Cuadrado	Cubo	Raíz cuadrada
1	1	1	1
2	4	8	1.414
3	9	27	1.732
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
9	81	729	3

13. Introducir el número de alumnos de un grupo y contabilizar por separado en qué sector de Guadalajara viven o si en su defecto viven en otro municipio; para tal efecto se desplegará un menú con las siguientes opciones :

R) Reforma

H) Hidalgo

L) Libertad

J) Juárez

O) Otro municipio

¿Dónde vives?

14. Suponer que se tiene un conjunto de notas finales de un grupo de 40 alumnos. Calcular e imprimir el promedio de las notas, la calificación final más alta y más baja de todo el grupo.
15. Imprimir una tabla de dos columnas para la conversión entre las temperaturas en grados Fahrenheit —comprendidas entre 10 °F y 200 °F, según incrementos de 15 °F— y su equivalente en grados Celsius. La conversión para grados Celsius es:

$$°C = \frac{5 * (°F - 32)}{9}$$

16. Calcular el promedio de los cuadrados de los números comprendidos entre 1 y un número *n* dado (inclusive). Es decir:

$$\text{promedio} = \frac{1^2 + 2^2 + \cdots + n^2}{n}$$

17. Para *n* términos, calcular las siguientes series (donde sea necesario leer el valor de x):

a) 1/2 + 3/4 + 7/8 +15/16...

b) 5/6 + 11/12 + 23/24,...

c) $(x - 1)^2 + (x - 3)^4 + (x - 5)^2 + (x - 7)^4 +$

d) $(x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} \ldots$

18. Leer dos números enteros (el primer número tendrá que ser mayor que el segundo) y calcular su producto mediante sumas sucesivas.
19. Leer un número entero positivo y averiguar si es perfecto. Un número es perfecto cuando es igual a la suma de sus divisores excepto él mismo. 28 es perfecto, porque 28 = 1 + 2 + 4 + 7 +14.

Ejemplos





140 = 1*2*2*5*7, MFP = 7; 13860 = 2*2*3*3*5*7*11, MFP = 11.

20. Leer una secuencia de *n* números y posteriormente determinar:

a) El mayor de los que son múltiplos de 5 y el menor de los que son múltiplos de 3

b) La suma de los pares y el producto de los que son múltiplo de 7

21. Leer la fecha de nacimiento de una persona (día, mes, año) e imprimir el número de días vividos por esa persona hasta la fecha actual. También calcular cuántos años bisiestos han pasado. El cálculo se hará sólo para personas que nacieron después del año 0, y antes del 2100.

Parte IV. Realizar la prueba de escritorio o corrida a mano de los siguientes programas:

Ejercicio 1

```
main()
{
```

Ejercicio 2

```
main()
{
```

(*continúa*)

(*continuación*)

```
   int i, j,x;
   i= j=1;
     for (x= 2; x<=15;x=x+3)

        {
          j+= i;
          i= j+x/2;
          printf ("%d, %d\n", i, j);
        }
   getch();
   return 0;
}
```

```
   int a=1, b=1, c;
   for (c=2;c<=10;c+=2)
   {
      if (a<10) {
                 a+=c;
                 b=c+b%2;
               }
            else b=c+a*2;
            a-=2;
            printf("%d,%d\n",a,b);
    }
    getch(); return  0;
}
```

Ejercicio 3

```
main()
{
   int i,j=1,k=1;
   for (i=10; i>1; i=i-2)
   {
     if (j+k<10) j=j*2+i;
           else j/=2;
       k=i%j;
       printf  ("%d,%d\n",j,k);
   }
   getch();
   return  0;
}
```

Ejercicio 4

```
main ()
{
   int x, y=3, z=3  ;
   for (x=3; x<13; x=x+2)
   {
      if (y>7)
      {
              y=z%x*2;  z+=4;
      }
      else {
            z=3*x+y;  y+=2;
            }
      printf("%d,%d\n",y, z);
   } getch(); return  0;
}
```

3.6.10 Ciclos anidados

En esta sección nos dedicaremos a los ciclos anidados, ya que anteriormente vimos los anidamientos con el *si (if)* y el *segun_sea (switch)*.

En un algoritmo puede haber varios bucles. Éstos pueden ser anidados o independientes.

Dentro de las instrucciones que se pueden incluir en un ciclo se encuentran, a su vez, otros ciclos, es decir, instrucciones *mientras (while)*, *hacer-mientras (do-while)*, y *desde (for)*. Este tipo de construcciones, en las que se incluye una instrucción de repetición dentro del cuerpo de otra, se conoce como *iteración anidada*.

En una iteración anidada, en cada iteración del ciclo exterior se completa un grupo de iteraciones del ciclo interior. Considere, por ejemplo, el siguiente fragmento de código:

```
for (i=1; i<=2; i++)
for (j=1; j<=3; j++)
printf("%d %d\n", i, j);
```

El ciclo exterior se ejecutará dos veces, haciendo variar la variable *i* por los valores 1, 2. En cada una de estas iteraciones se ejecutará un ciclo de tres iteraciones (el *desde (for)* interior), en el que se variará la variable *j* por los valores 1, 2, 3. De esta forma, por cada valor que toma la variable *i*, la variable *j* tomará todos sus valores. Así, la instrucción *printf* se ejecutará seis veces (2×3); de la misma manera lo hicimos en el ejemplo 10.

Es posible anidar cualquier cantidad de instrucciones de iteración, prácticamente hasta que la implementación del compilador lo soporte. Sin embargo, demasiados niveles de anidamiento pueden volver el programa difícil de leer.

3.6.11 Ejercicios resueltos utilizando ciclos anidados

Ejercicio 1. Elevar un número X a una potencia Y, preguntándole al usuario si desea realizar el cálculo de otra potencia.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero cont, x, y, res; caracter resp← 'S'	`int cont,x,y,res; char resp='S';`
mientras (resp = 'S')	`while ( resp == 'S')`
inicio	`{`
res←1	`res=1;`
imprimir "Número que se elevará a una potencia:"	`printf("No. que se elevará a una potencia:  ");`
leer x	`scanf("%d",&x);`
imprimir "A qué potencia elevará el número"	`printf("A qué potencia elevará el número:  ");`
leer y	`scanf("%d",&y);`
desde (cont←1, cont<=y, cont←cont+1)	`for (cont=1; cont<=y ; cont=cont+ 1)`
res←res * x	`res=res  * x;`
imprimir "La potencia es: ",res	`printf("La potencia es: %d\n", res);`
imprimir " desea realizar otra potencia S / N"	`printf(" desea realizar otra potencia S/N  ");`
leer resp	`scanf("%s",&resp);`
fin	`}`
fin	`}`

El ejercicio 1 utiliza dos estructuras de control repetitivas: el *mientras* para saber si desea realizar otra potencia, y el *desde* para elevar un número a una potencia. La lógica para elevar un número *X* a una potencia y se explica en el ejercicio 7, así que nos centraremos en la explicación de la estructura repetitiva *mientras*. Se incorpora una variable centinela llamada *resp*, la cual se declara de tipo caracter ya que almacenará una *S* si desea realizar otra potencia y una *N* si desea terminar el programa. Se inicializa la variable *resp* = *'S'* para que entre por lo menos una vez a realizar el cálculo de la potencia; al terminar el ciclo *desde* se sale de éste y se imprime el resultado del primer cálculo.

Ejemplos



Si $x = 2$ y $y = 4$, imprime *La potencia es: 16*, y se le pregunta al usuario si desea realizar otra vez el programa con la instrucción *imprimir "desea realizar otra potencia S / N"* y se espera la respuesta del usuario, la cual se almacena en la variable *resp* con la instrucción *leer resp*; la siguiente instrucción es el fin del ciclo *mientras*, la cual regresa a revisar la condición del *mientras (resp = `S`)*, si el usuario eligió *S*, vuelve a solicitarse datos de la base y la potencia y se realiza otra vez el cálculo de la potencia, pero si eligió *N*, se sale del ciclo *mientras* y se termina el algoritmo y programa. Cabe mencionar que se inicia nuevamente la variable *res←1*, ya que de lo contrario se queda la variable *res* con el valor que se haya tomado de la corrida anterior, en este caso almacena el *res = 16* y el resultado sería incorrecto.

Una posible salida en pantalla, después de ejecutar el programa sería:

Escriba el número que se elevará a una potencia: 3.

A qué potencia elevará el número: 3.

El 3 elevado a la potencia 3 es: 27.

desea realizar otra potencia S / N S.

Escriba el número que se elevará a una potencia: 2.

A qué potencia elevará el número: 4.

La potencia es: 16.

desea realizar otra potencia S / N N.

Ejercicio 2. Calcular el factorial de un número entero y preguntar al usuario si desea realizar otra vez el cálculo de un factorial.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`int main()`
inicio	`{`
entero i, x	`int i, x;`
enterolargo fact	`long fact;`
caracter resp	`char resp;`
hacer	`do`
inicio	`{`
fact ←1	`fact =1;`
imprimir " Dame un número"	`printf("Dame un número:  ");`
leer x	`scanf("%d",&x);`
desde (i← x, i > 1, i ← i − 1)	`for (i= x; i > 1; i--)`
fact←fact * i	`fact=fact * i;`
imprimir " El factorial de", x ," = ", fact	`printf("El factorial de %d = %ld\n",x,fact);`
imprimir "Deseas realizar otro cálculo S/N"	`printf("Deseas realizar otro cálculo S/N ");`
leer resp	`scanf("%s",&resp);`
fin	`}`
mientras (resp = `S`)	`while  (resp =='S');`
	`getch();`
	`return  0;`
fin	`}`

El ejercicio 2 utiliza dos estructuras de control repetitivas el *hacer_mientras* para saber si desea realizar otro factorial, y el *desde* para realizar el cálculo del factorial. La lógica para calcular el factorial se explica en el ejercicio 8 de la estructura *desde*, así que nos centraremos en la explicación de la estructura repetitiva *hacer_ mientras*. Se incorpora una variable *resp*, la cual se declara de tipo caracter ya que almacenará una *S* si desea realizar otro factorial y una *N* si quiere terminar el programa. Se inicia el programa y se pregunta *el número para calcular su factorial?*, se realiza el cálculo, se sale del ciclo *desde* y *se* imprime *El factorial de 5 es 120*, se le pregunta al usuario si desea realizar otra vez el programa con la instrucción *imprimir " Desea realizar otro cálculo del factorial S / N"* y se espera la respuesta del usuario, la cual se almacena en la variable *resp* con la instrucción *leer resp*. La siguiente instrucción es el fin del ciclo *hacer_mientras* y la siguiente instrucción *mientras (resp = `S`)*, si el usuario eligió *S*, vuelven a solicitarse datos para calcular el factorial y se realiza otra vez el cálculo, pero si escogió *N*, se sale del ciclo *mientras* y se termina el programa. Cabe mencionar que si eligió *S* se inicia nuevamente la variable *fact←1*, ya que de lo contrario se queda la variable *fact* con el valor que se haya tomado de la corrida anterior, es decir *fact= 120* y el resultado sería incorrecto.

Ejercicio 3. Realizar las tablas de multiplicar del 1 al 10 hasta el múltiplo 15.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero i, j	`int i, j;`
desde (i← 1, i <= 10, i ← i + 1)	`for ( i=1;i<= 10;i++)`
inicio	`{`
imprimir " La tabla del número ", i , " es:"	`printf("La tabla del número %d es:\n",i);`
desde (j ← 1, j <= 15, j ← j + 1)	`for (j = 1; j <= 15; j++)`
imprimir i, " * " , j , " =", i * j	`printf("%d * %d = %d\n",i,j,i*j);`
	`getch(); return 0;`
fin	`}`
fin	`}`

Para poder imprimir las 10 tablas de multiplicar se necesitan dos ciclos *desde* anidados, el primer ciclo *desde (i← 1, i <= 10, i ← i + 1)* para cada tabla, ya que a la variable *i* se le asigna 1 para que se imprima la tabla del 1, *i* tomará el valor de 2 para imprimir la tabla del 2 y así sucesivamente hasta que *i* tome el valor de 10 e imprima la última tabla. El segundo *desde* sirve para imprimir los 15 múltiplos de cada tabla. Dentro del primer ciclo *desde*, la primera instrucción es imprimir "*la tabla del número*", i, "es:", para que nos muestre en pantalla "la tabla del número 1 es:", y la siguiente instrucción es *desde (j ← 1, j <= 15, j ← j + 1),* éste utiliza un contador *j* para cada múltiplo e inicializa con 1, para multiplicar e imprimir el resultado de 1 * 1 = 1, se incrementa *j* en 1, para multiplicar e imprimir el resultado de 1 * 2 = 2, se incrementa j en 1, para multiplicar e imprimir el resultado de 1 * 3 = 3, así sucesivamente hasta que j tome el valor de 15 imprima el resultado de 1 * 15 = 15; como la variable *j* ya llegó a su valor final *j <= 15*, se sale del ciclo *desde* interno y regresa al ciclo *desde* afuera, donde *i* toma el valor de 2, y se repite todo el ciclo de adentro para i = 2 "la tabla del número 2 es:"; la siguiente instrucción es *desde (j ← 1, j <= 15, j ← j + 1),* donde j inicializa otra vez en uno, para multiplicar e imprimir el resultado de 2 * 1 = 2, se incrementa j en 1 para multiplicar e imprimir el resultado de 2 * 2 = 4, se incrementa *j* en 1 para multiplicar e imprimir el resultado de 2 * 3 = 6, así sucesivamente hasta que *j* tome el valor de 15 e imprima el resultado de 2 * 15 = 30; como la variable *j* ya llegó a su valor final *j <= 15*, se sale del ciclo *desde* interno, y regresa al *desde* externo. Estas operaciones las hará hasta haber impreso las 10 tablas de multiplicar, es decir que se imprima la tabla del 10 hasta el múltiplo 15.

Las instrucciones del *desde i* necesitaron inicio y fin, ya que como vimos ejecuta dos instrucciones: el *imprimir* y el *desde*. El segundo *desde* (anidado) no necesita inicio y fin ya que sólo ejecuta una instrucción.

Ejercicio 4. Imprimir un triángulo de asteriscos *n* filas.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero i,j,nfila	`int i,j,nfila;`
imprimir "Número de filas del triángulo :"	`printf("Número de filas del triángulo:");`
leer nfila	`scanf("%d",&nfila);`
desde (i ←1, i <= nfila, i←i + 1)	`for(i=1;i<=nfila;i++)`
inicio	`{`
desde (j ←1, j<= i, j←j +1)	`for (j=1;j<=i;j++)`
imprimir "* "	`printf ("*  ");`
salto_linea	`printf ("\n");`
fin	`}`
	`getch(); return 0;`
fin	`}`

Una posible salida en pantalla, después de ejecutar el programa sería:

Número de filas del triángulo: 4

```
*
**
***
****
```

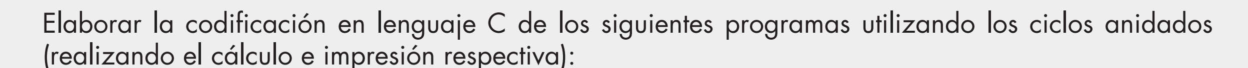
En el ejercicio 4, tenemos dos ciclos anidados *desde (for)*. Declaramos tres viables enteras: *i*, *j*, *nfila*; en seguida se pide el número de filas (en nuestro ejemplo capturamos 4). Para entender el proceso en los ciclos anidados haremos la corrida a mano:

i	j	imprimir "*" printf ("* ");	salto_linea printf ("\n");
1	1	*	No
1	2	Se rompe	Sí
2	1	*	No
2	2	* *	No
2	3	Se rompe	Sí
3	1	*	No
3	2	* *	No
3	3	* * *	No
3	4	Se rompe	Sí
4	1	*	No
4	2	* *	No
4	3	* * *	No
4	4	* * * *	No
4	5	Se rompe	Sí

Como podemos apreciar la variable *i* inicia con 1 hasta llegar a total de filas (*nfila*), ya que controla los renglones. La variable *j* inicia con 1 hasta llegar el valor de la variable *i*, ya que controla las columnas. Cada vez que se rompe el ciclo *j* salta renglón imprimiendo los asteriscos en el siguiente renglón.

En este ejemplo es necesario que el ciclo para la variable *i* lleve inicio-fin ({-}), debido a que tiene dos instrucciones: el *for* para la variable *j* y el salto de línea.

Ejercicios complementarios utilizando ciclos anidados



Elaborar la codificación en lenguaje C de los siguientes programas utilizando los ciclos anidados (realizando el cálculo e impresión respectiva):

1. Simular el comportamiento de un reloj digital, escribiendo la hora, minutos y segundos de un día desde las 0:00:00 horas hasta las 23:59:59 horas
2. Calcular los primeros 20 números primos.
3. Imprimir todos los números primos entre 2 números ingresados por el teclado. Ejemplo: Los números primos entre 5 y 15 son 5, 7, 11, 13.
4. Generar los 4 primeros números perfectos.
5. Generar de los primeros 40 números (del 1 al 40) sus factores primos.
6. Imprimir el abecedario en forma inversa es decir de la z a la a y luego vaya eliminando de una letra en una empezando por la z hasta que quede la a:

   zywvtsrqponmlkjihgfedcba

   ywvtsrqponmlkjihgfedcba

   wvtsrqponmlkjihgfedcba

   …

   rqponmlkjihgfedcba

   qponmlkjihgfedcba

   ponmlkjihgfedcba

   …

   edcba

   dcba

   cba

   ba

   a

7. Leer un carácter y dependiendo del carácter leído obtener una pirámide con el alfabeto.

   Ejemplos 

   si el carácter fue 'c', se imprimirá:

   A

   Bb

   Ccc

8. Leer un nombre y con él llene toda la pantalla, luego, solicite las coordenadas fila1, columna1 (F1,C1) y fila2, columna2 (F2,C2), en ese rectángulo que se forma con las coordenadas, se debe limpiar sólo esa parte de la pantalla.

9. Leer un número e imprimir un triángulo de asteriscos.

Ejemplos



Si N=5, imprimirá:

```
*
* * *
* * * * *
* * * * * * *
```

10. Leer un número e imprimir los triángulos de asteriscos de la anchura que especifica el número.

Ejemplos





Si *n* es 5, los triángulos serían:

```
*             * * * * *       * * * * *             *
* *             * * * *       * * * *             * *
* * *             * * *       * * *             * * *
* * * *             * *       * *             * * * *
* * * * *             *       *             * * * * *
```

11. Leer el ancho (número impar) de un rombo e imprimirlo con asteriscos. Si el ancho = 7.

```
      *
    * * *
  * * * * *
* * * * * * *
  * * * * *
    * * *
      *
```

12. Leer un número e imprimir una pirámide con los números consecutivos.

Ejemplos



Si n = 16:

```
1
2  3
4  5  6
7  8  9  10
11 12 13 14 15
16
```

13. Leer un número e imprimir una pirámide de dígitos.

Ejemplos



Si N = 5, imprimirá:

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 2 3 1
```

14. Leer un número e imprimir una pirámide de dígitos.

Ejemplos



Si N = 5, imprimirá:

```
1
2 3 2
3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

15. Leer un número e imprimir los rombos de la anchura que especifica el número.

Ejemplos



Si n = 4, los rombos serían:

```
1          *
1 2        * *
1 2 3      * * *
1 2 3 4    * * * *
1 2 3      * * *
1 2        * *
1          *
```

16. Leer un número mayor que 0 y menor que 10 e imprimir en pantalla triángulos de base igual al número introducido y dispuestos como se muestran en las siguientes figuras.

Ejemplos



Introduzca un número entre 0 y 10: 4.

```
      1          4 4 4 4 4 4 4
    2 2 2          3 3 3 3 3
  3 3 3 3 3          2 2 2
4 4 4 4 4 4 4          1
  3 3 3 3 3          2 2 2
    2 2 2          3 3 3 3 3
      1          4 4 4 4 4 4 4
```

17. Imprimir en una pantalla las 10 tablas de multiplicar.
18. Leer 10 números e imprimir la suma de los que sean primos.
19. Leer un número positivo e imprimir las sumas de números enteros positivos consecutivos que den el número introducido. 50 = 8 + 9 + 10 + 11 + 12; 50 = 11 + 12 + 13 + 14.
20. Leer un número y calcular su factorización. Al final imprimir el máximo factor primo (MFP) obtenido. Se entiende por factorización de un número, el conjunto de números primos por el que es divisible, incluyendo aquellos que se repiten.



Capítulo 4

Arreglos

Al término de este capítulo, el alumno será capaz de

- Manejar variables que almacenan más de un valor e identificará en qué casos utilizar los arreglos unidimensionales y bidimensionales para la resolución de problemas.

Contenido



4.1 Definición

Un arreglo es un tipo de dato estructurado que almacena en una sola variable un conjunto limitado de datos o elementos del mismo tipo. Asimismo, es un conjunto de localidades de memoria contiguas donde la dirección más baja corresponde al primer elemento y la dirección más alta al último. Por sí mismo, el nombre del arreglo apunta a la dirección del primer elemento del arreglo. Los datos se llaman elementos del arreglo y su posición se numera consecutivamente: 1, 2, 3…*n*. Un arreglo en lenguaje C inicia en la posición cero, por lo tanto el *i-ésimo* elemento está en la posición *i*-1, es decir si el arreglo llamado *a* tiene *n* elementos, sus nombres son *a[0], a[1], ..., a[n-1]*. El tipo de elementos almacenados en el arreglo puede ser cualquier tipo de dato.

Para acceder a un elemento específico de un arreglo se usa un índice o subíndice.

Un *arreglo* se caracteriza por:

1. Ser una lista de un número finito de *n* elementos del mismo tipo.
2. Almacenar los elementos del arreglo en memoria contigua.
3. Tener un único nombre de variable que representa a todos los elementos y éstos se diferencian por un índice o subíndice.
4. Acceder de manera directa o aleatoria a los elementos individuales del arreglo, por el nombre del arreglo y el índice o subíndice.

La importancia de declarar arreglos de tamaño adecuado

Al igual que cualquier variable, los arreglos ocupan espacio en memoria. El programador especifica el tipo de dato y el total de elementos requerido por el arreglo de tal forma que la computadora pueda reservar la cantidad apropiada de memoria. Si el programador declara un arreglo de 100 elementos de tipo entero y sólo utiliza 10 espacios, desperdicia 90 en memoria para datos de tipo entero. Por lo contrario, si se declara un arreglo de 50 elementos y se quieren manejar 100, faltarán 50 espacios; sin embargo, no se presentará mensaje de error en el tiempo de compilación o ejecución, sino hasta que el sistema operativo se dé cuenta y por lo tanto surja la falla en el programa.

Se pueden declarar varios arreglos en una sola instrucción y de esta forma reservar la memoria necesaria. Para reservar 100 elementos para el arreglo *a* y 50 elementos para el arreglo *x*, ambos de tipo entero, se puede utilizar la siguiente declaración:

Pseudocódigo	Lenguaje C
entero a[100], x[50]	`int a[100], x[50];`

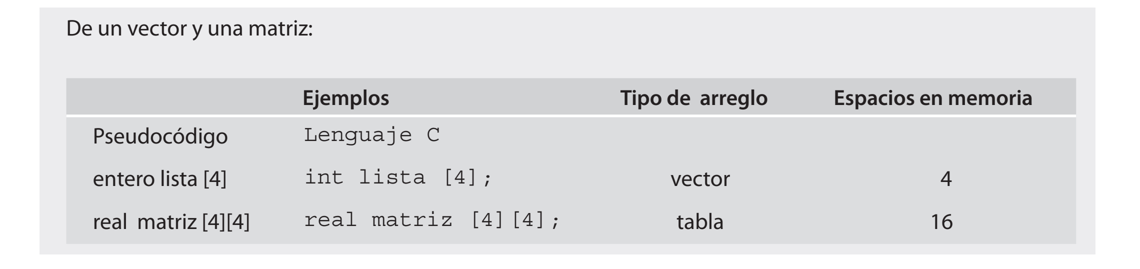
Los arreglos se clasifican en:

Unidimensionales (vectores o listas)
Bidimensionales (tablas o matrices)
Multidimensionales (más de dos dimensiones)

Los más utilizados son los *unidimensionales* y los *bidimensionales*; a continuación se describirá cada uno de ellos.

Ejemplos





De un vector y una matriz:

	Ejemplos	Tipo de arreglo	Espacios en memoria
Pseudocódigo	`Lenguaje C`		
entero lista [4]	`int lista [4];`	vector	4
real matriz [4][4]	`real matriz [4][4];`	tabla	16

4.2 Arreglos unidimensionales (vectores o listas)

Un arreglo unidimensional es un conjunto de *n* elementos del mismo tipo almacenados en memoria continua en un vector o lista. Para acceder a cada elemento del arreglo se requiere de un solo índice o subíndice, el cual representa la posición en la que se encuentra.

Formato para declarar un arreglo unidimensional

Pseudocódigo	Lenguaje C
tipo_dato identif_arreglo[tam_arreglo]	`tipo_dato identif_arreglo[tam_arreglo];`

Donde:

tipo_dato se refiere al tipo de dato de cada elemento del arreglo; puede ser entero, real, carácter, etcétera.
identif_arreglo es el nombre que representa a todo el arreglo
tam_arreglo es la cantidad de elementos que contiene el arreglo.

Si tomamos la declaración del arreglo *lista* del ejemplo anterior, así se representaría en memoria *entero lista* [4]:

Posición de la memoria	1000	1001	1002	1003	1004	1005	1006	1007
lista	0		1		2		3	

Los enteros requieren de dos bytes para almacenarse en memoria; como se muestra, por cada posición se requiere de dos localidades de memoria, por ejemplo el 0 ocupa la posición 1000 y 1001.

La cantidad de arreglos que se pueden declarar dependerá de la memoria libre, que comúnmente es de 64 Kbytes; esta cantidad puede variar e incluso se puede utilizar más memoria disponible siempre y cuando la computadora cuente con ella.

A continuación se muestra un arreglo de números reales cuyo identificador es *x*:

Pseudocódigo	Lenguaje C
real x[8]	`float x[8];`

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	← posiciones
elementos →	4.2	12.0	3.45	4.32	0.31	51.9	2.65	13.0	

Este arreglo contiene ocho elementos almacenados entre la posición (0-7). Para referirnos a un elemento en particular dentro del arreglo, especificamos el nombre del arreglo y el número de posición donde se encuentra ubicado. La posición del arreglo va entre paréntesis cuadrados o corchetes ("[ ]") para el lenguaje C; según el lenguaje de programación será la sintaxis requerida.

El primer elemento en un arreglo es almacenado en la posición cero para el lenguaje C y en la posición uno para el lenguaje Pascal. En este caso trabajaremos el pseudocódigo en C. Por lo tanto el primer elemento de un arreglo *x* se encuentra en la posición cero y se conoce como *x* [0], el segundo como *x* [1], el séptimo como *x* [6] y en general, el elemento de orden *i* del arreglo *x* será *x* [*i*-1], ya que se encuentra en la posición *i*–1, donde *i* es un subíndice que sirve para hacer referencia a la posición en el arreglo. Es decir, si almacenamos en un arreglo las edades de los alumnos de un salón de clases (40), la edad del primer alumno estará almacenada en la posición 0, la del segundo en la posición 1 y así sucesivamente hasta la del cuadragésimo alumno en la posición 39. Los identificadores de los arreglos deben tener las características de los identificadores del lenguaje.

Si la instrucción en pseudocódigo fuera imprimir *x*[4] se mostrará el valor de 0.31.

Si se requiere guardar en una posición específica del arreglo se debe escribir el identificador del arreglo y su posición, por ejemplo: leer *x*[0]. Para llenar un arreglo completo se utiliza generalmente el ciclo *desde (for)* facilitando con la variable de control el incremento de la *i*, donde la *i* representa el subíndice.

Para imprimir la suma de los valores contenidos en los primeros tres elementos del arreglo *x*, escribiríamos:

Pseudocódigo	Lenguaje C
a ← x[0] + x[1] + x[2]	`a = x[0] +x[1] +x[2];`
imprimir a	`printf("%f",a);`

Donde la variable *a* es de tipo real, ya que los elementos del arreglo *x* son de ese tipo.

Para dividir el valor del séptimo elemento del arreglo *x* entre 2 y asignar el resultado a la variable *c* escribiríamos:

Pseudocódigo	Lenguaje C
c← x[6] / 2	`c = x[6]/2;`

Nota: Trabajar con arreglos es similar a trabajar con variables; la diferencia es que se necesita escribir el nombre o identificador de la variable que representa todo el arreglo, para este caso *x* y además el subíndice *i*. Por ejemplo, si se quiere imprimir el valor del quinto elemento, éste se encuentra ubicado en la posición 4; la instrucción en pseudocódigo es imprimir *x[i]* y scanf ("%d",&*x[i]*) en lenguaje C, donde *x* es el arreglo e *i* la posición, que en este caso es 4.

Un subíndice debe ser un entero o una expresión cuyo resultado sea entero, por ejemplo *a[2.5]* **no** es válido. Si un programa utiliza una expresión como subíndice, entonces la expresión se evalúa para determinar el subíndice.

Ejemplos



Si i = 2 y *j* = 4, entonces el enunciado:

Pseudocódigo	Lenguaje C
c[i + j]←10	`c [i + j] = 10;`

almacena el valor de 10 en el arreglo *c* en la posición *[6]*. Nótese que el nombre de arreglo con subíndice *i* se utiliza al lado izquierdo de la asignación, como si c[i + j] fuera una variable.

Examinaremos el *arreglo x*. Sus ocho elementos se conocen como *x[0], x[1], x[2], ..., x[7]*. El valor de *x[0]* es 4.2, el valor de *x[2]* es 3.45, y así sucesivamente hasta el valor de *x[7]*, que es 13.0.

Se pueden asignar valores a los elementos del arreglo antes de utilizarlos tal como se asignan valores a variables. Una manera es inicializando el arreglo y la otra leyendo cada elemento del mismo. Comenzaremos con la primera posibilidad.

4.2.1 Inicialización de arreglos unidimensionales

En el momento de declarar el arreglo, se especifican los valores.

Sintaxis:

Pseudocódigo	Lenguaje C
tipo_dato identif [tam_arreglo]←{valores}	`tipo_dato identif[tam_arreglo]={valores};`
entero lista [5] ← { 10,17,8,4,9 }	`int lista [5] = {10,17,8,4,9};`

La asignación de los valores se realiza al declarar el arreglo mediante el operador de asignación (←/=) y los valores contenidos dentro de las llaves { } a cada posición del arreglo; los valores dentro de las llaves se

deben separar por una coma (,) y no es necesario asignarle un valor a cada posición ya que el compilador del lenguaje C se encargará de hacerlo según el lugar donde se haya declarado el arreglo, comenzando por la posición cero. Este modo de asignación no es válido si lo hacemos después de declarar el arreglo. Si el número de valores es menor al tamaño del arreglo a las posiciones faltantes les asigna *cero*.

4.2.2 Lectura e impresión de un arreglo unidimensional

La declaración de arreglos se hace al mismo tiempo que la declaración de variables normales, como se mencionó anteriormente:

Pseudocódigo	Lenguaje C
tipo_dato identif [tam_arreglo]	`tipo_dato identif [tam_arreglo];`
entero lista [10]	`int lista [10];`

Es fácil procesar los elementos de un arreglo mediante ciclos repetitivos porque facilitan la lectura, impresión, consulta y modificación de todos los elementos del arreglo, reduciendo el trabajo a unas cuantas líneas bien empleadas. Cuando la computadora ejecuta un ciclo y la instrucción *leer (scanf)* el programa almacena los valores en la variable arreglo. Para utilizar ciclos repetitivos es necesario el uso de variables de apoyo; el siguiente ejemplo muestra la forma de pedirle 10 números al usuario e imprimirlos después, utilizando ciclos repetitivos *for*. Veamos cómo podemos leer e imprimir el arreglo anterior:

Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
principal ()	main()
inicio	{
entero lista[10], i	int lista[10], i;
desde (i← 0, i <=9 ,i←i+1)	for (i=0; i<=9;i++)
inicio	{
imprimir "Dame el elemento"	printf("Dame el elemento");
leer lista[i]	scanf("%d",&lista[i]);
fin	}
imprimir "Elementos de la lista"	printf("Elementos de la lista\n");
desde (i←0, i<=9, i←i+1)	for (i=0; i<=9;i++)
imprimir lista[i]	printf("%d\n",lista[i]);
	getch();
	return 0;
fin	}

El programa utiliza la estructura de repetición *desde (for)* y *leer (scanf)* para capturar los elementos del arreglo llamado lista, e imprime el arreglo mediante otro *desde (for)* y el *imprimir (printf)*, donde el *desde* recorre todas las posiciones del arreglo. Es importante señalar que se debe respetar el tipo de dato del arreglo y además especificar la posición del arreglo en la que se quiere guardar el valor.

La manera de declarar el tamaño del arreglo varía en cada compilador. Por ejemplo en Turbo C de Borland, el tamaño debe ser constante como en el ejemplo presentado, donde el máximo tamaño posible es de 30 elementos y el que realmente ocuparemos es *n*.

El programa podría quedar abierto si manejamos un tamaño de arreglo *n*:

Pseudocódigo	Lenguaje C (Turbo C, DEV-CPP y Code::Blocks)
constante MAX←30	`#define MAX 30`
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main ()`
inicio	`{`
entero lista[MAX], i,n	`int lista[MAX], i,n;`
imprimir "Dame el tamaño del arreglo:"	`printf ("Dame el tamaño del arreglo: ");`
leer n	`scanf ("%d",&n);`
desde (i←0, i<n , i←i+1)	`for (i=0; i<n; i++)`
inicio	`{`
imprimir "Dame el elemento ",i	`printf("Dame el elemento %d ",i );`
leer lista[i]	`scanf("%d",&lista[i]);`
fin	`}`
imprimir("Elementos de la lista")	`printf("Elementos de la lista\n");`
desde (i←0, i<n , i←i+1)	`for (i=0; i<n;i++)`
imprimir lista[i]	`printf("%d\n",lista[i]);`
	`getch();`
	`return 0;`
fin	`}`

En el caso de compiladores como DEV-CPP y Code::Blocks tenemos la ventaja de manejar un tamaño variable, ya que primero leemos *n* y posteriormente declaramos el arreglo respectivo. En Turbo C de Borland nos marcaría el error: *Constant expression required*.

Ejemplos



DEV-CPP y Code::Blocks

```
#include <stdio.h>
#include <conio.h>
main ()
{
     int i,n;
     printf ("Dame el tamaño del arreglo: ");
     scanf ("%d",&n);
     int lista[n];
     for (i=0; i<n;i++)
     {
         printf("Dame el elemento %d ",i );
         scanf("%d",&lista[i]);
     }
     printf("Elementos de la lista\n");
     for (i=0; i<n;i++)
         printf("%d\n",lista[i] );
     getch();
     return 0;
}
```

4.2.3 Modificación de un elemento del arreglo unidimensional

Podemos modificar los elementos de un vector en cualquier momento, sólo es necesario especificar el nombre del arreglo unidimensional, la posición y el nuevo valor. Enseguida se muestra la sintaxis a seguir:

Pseudocódigo	Lenguaje C
tipo_dato ident_arr [pos] ← valor	`tipo_dato ident_arr[pos]=valor;`
entero b[3] ← 18	`int b[3]=18;`

Donde *valor* es un dato, el resultado de una llamada a función o de alguna operación lógica o aritmética, etc. En este ejemplo se le asigna el valor 18 al cuarto elemento del arreglo que se encuentra en la posición 3.

4.2.4 Arreglos de caracteres

Los arreglos son necesarios para la implementación de cadenas de caracteres. Una cadena de texto es un conjunto de caracteres. Hay que recordar que en el lenguaje C no existe el tipo de dato cadena (string) como en otros lenguajes de programación, por lo que se utiliza un arreglo de caracteres, para poder almacenar una cadena:

Ejemplos



Pseudocódigo	Lenguaje C
caracter cad[ ] ←"Lenguaje"	`char cad[ ]="Lenguaje";`

Una cadena de caracteres es un arreglo de caracteres que contiene al final el carácter nulo (\0); por esta razón es necesario que al declarar los arreglos éstos sean de un carácter más que la cadena más grande. El compilador inserta automáticamente un carácter nulo al final de la cadena, de modo que la secuencia real sería:

Pseudocódigo	Lenguaje C
caracter cad[9] ←"Lenguaje"	`char cad[9]="Lenguaje";`

La cadena quedaría almacenada como sigue:

Posición memoria	*1315*	*1316*	*1317*	*1318*	*1319*	*1320*	*1321*	*1322*	*1323*
Contenido	L	e	n	g	u	a	j	e	\0
Elemento del arreglo	cad[0]	cad[1]	cad[2]	cad[3]	cad[4]	cad[5]	cad[6]	cad[7]	cad[8]

Una opción para almacenar una cadena de caracteres es el uso de la palabra reservada *scanf* (variable) pero, si queremos almacenar una cadena con espacios en blanco no lo podemos hacer con ella, sino que debemos utilizar la palabra reservada *gets*, que se encuentra dentro de la librería *string.h; gets* sólo se utiliza para leer cadenas de caracteres y *scanf* para leer cualquier tipo de variable, de preferencia de tipo numérico (vea detalles en el capítulo de programación modular y anexo A).

gets

Introduce una cadena de caracteres del teclado hasta que se encuentra un carácter '\n" (nueva línea), dicho carácter no se añade a la cadena. Se agrega al final del arreglo un carácter de terminación NULL.

Sintaxis: gets(variable_cadena);

scanf

Lee una cadena. El argumento correspondiente es un apuntador a un arreglo del tipo char, que es lo suficiente extenso para contener la cadena y un carácter de terminación NULL.

Sintaxis: scanf (“%s”, &variable_cadena);

Cuando inicializamos una cadena de caracteres no es necesario definir el tamaño, el compilador siempre añade un carácter nulo al final.

Ejemplos



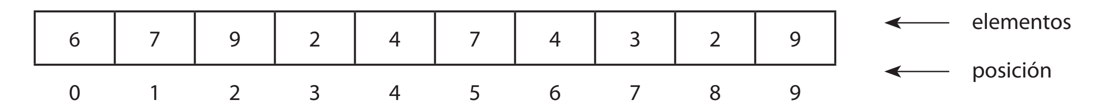
```
char cadena[ ] = “Programación” //cadena de 13 caracteres
```

4.2.5 Ejercicios resueltos de arreglos unidimensionales

Ejercicio 1. Inicialice un arreglo de 10 elementos, realice e imprima la sumatoria.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero x[10] ← {6,7,9,2,4,7,4,3,2,9}	`int x[10]={6,7,9,2,4,7,4,3,2,9};`
entero i, s←0	`int i,s=0;`
desde (i ← 0, i < 10, i←i+1)	`for (i=0;i<10;i++)`
s ← s + x [i]	`s+=x[i];`
imprimir “ La suma es”, s	`printf(“La suma es %d”,s);`
	`getch(); return 0;`
fin	`}`

En el ejercicio 1, en el momento de declarar la variable *x*, que es de tipo arreglo, se inicializa el arreglo colocando los valores entre llaves: {6,7,9,2,4,7,4,3,2,9}, y son de tipo entero; no hay que olvidar que la posición inicial es *cero*, por lo tanto:



x[0]=6, x[1]=7, x[2]=9, ... x[9]=9, donde x[10] no existe. La variable *i* se utiliza dentro del ciclo *desde (for)* y es la manera más sencilla de almacenar o imprimir datos en un arreglo, por la facilidad de incrementar o decrementar la variable *i*, y de esta manera se vaya recorriendo el arreglo. Dentro del ciclo *desde (for)*, la variable *s* va a ir acumulando la suma de todos los elementos del arreglo hasta llegar al décimo elemento, que se encuentra situado en la posición nueve y en este momento se sale del ciclo *desde (for)* para imprimir el resultado de la suma.

Recuerde que la expresión s+=x[i] en lenguaje C es equivalente a s=s+x[i]; hay que tener cuidado de dejar un espacio después de s+ *(s+ = x[i])*, ya que se producirá un error durante la compilación. La variable *s* acumularía: 6 + 7 + 9 + 2 + 4 + 7 + 4 + 3 + 2 + 9 = 53.

En el ejercicio 2 la variable de tipo arreglo *alumn* almacenará la edad de un máximo de 45 alumnos y los datos serán de tipo entero, ya que en la declaración se reserva este espacio en memoria. Luego se pregunta el *Total de alumnos* de ese salón de clases y se guarda en la variable *total_a*, que le sirve al ciclo *desde (for)*

para saber cuántas veces entrará; si el *total_a* es igual a 20 solamente entrará y se almacenarán 20 edades de la posición [0 a la 19], la etiqueta que se encuentra dentro del ciclo desde (for) *"Escriba la edad del alumno: ",i+1* imprimirá el valor de $i + 1$ ya que la variable *i* inicia en la posición 0 y si queremos preguntar por la edad del primer alumno tenemos que incrementarle 1 a la variable *i* para que pueda imprimir *Escriba la edad del alumno: 1* y así sucesivamente. Ya almacenadas las edades se le pregunta al usuario de qué alumno desea ver su edad y se almacena en la variable *i*, para luego mostrar el valor almacenado en esa posición menos uno, ya que si se desea ver la edad del alumno 5 estará almacenada en la posición 4.

Ejercicio 2. Almacene la edad de todos los alumnos de un salón de clases y posteriormente preguntar e imprimir la edad de alguno de ellos.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero alumn[45], total_a , i	`int alumn[45], total_a,i;`
imprimir "Total de alumnos:"	`printf("Total de alumnos:\n");`
leer total_a	`scanf("%d",&total_a);`
desde (i←0,i< total_a, i←i+1)	`for( i=0; i< total_a; i++)`
inicio	`{`
imprimir "Dame la edad del alumno: ",i+1	`printf("Dame la edad del alumno: %d\n",i+1);`
leer alumn[i]	`scanf("%d", &alumn[i]);`
fin	`}`
imprimir "¿De qué alumno quieres su edad?"	`printf("¿De qué alumno quieres su edad?");`
leer i	`scanf("%d",&i);`
imprimir "La edad",i ,"es :", alumn[i−1]	`printf("La edad es: %d", alumn[i−1]);`
	`getch(); return 0;`
fin	`}`

Ejemplos



En *total_a = 6* la variable *i* inicializa en 0 y las edades se irán almacenando en *alumn[0]=20, alumn[1] =18, alumn[2] =22, alumn[3] =19, alumn[4] =21, alumn[5] =20;* nótese que el cambio del índice *alumn[i]* se debe al cambio de la variable *i* dentro del ciclo *desde (for)*. Cuando se pregunta *¿De qué alumno deseas ver su edad?*, si se elige *i =4*, el programa imprimirá *La edad del alumno 4 es: 19*, que realmente se encuentra ubicado en la posición 3, ya que la edad correcta la encontramos en *alumn[i-1]*.

En el ejercicio 3 se utiliza el arreglo *cal*, que es de tamaño 10 y los datos que se almacenan son de tipo reales ya que las calificaciones pueden ser enteras o reales y cuando mucho el alumno pudo cursar 9 materias en un mismo semestre. La última posición se deja para almacenar el promedio. Se pregunta *¿Cuántas materias llevas (max 9)?* y el dato se almacena en la variable *mat*, luego entrará al ciclo *desde (for)*, donde la variable *i* empezará en 0 y llegará a uno menos de *mat* por el signo "<" y por cada vez que entre al ciclo *desde (for)* se irá almacenando en *cal[i]* y sumando la calificación en *prom*; para luego cuando se salga del ciclo *desde (for)* llegue al número de materias cursadas; se puede almacenar en la siguiente posición el promedio que se calcula de la suma de todas las calificaciones *prom* dividido entre el total de materias cursadas *mat*. La posición siempre será *mat* ya que si el alumno cursó seis materias y sus calificaciones estarán almacenadas de la posición 0 a la 5 y queda disponible la posición 6 para guardar el promedio.

Ejercicio 3. Calcule el promedio de las calificaciones de las materias del semestre anterior, utilizando un arreglo para almacenar todas las calificaciones y el promedio guárdelo en la siguiente posición después de la última calificación:

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
real cal[10], prom←0	`float cal[10],prom=0;`
entero i , mat	`int i,mat;`
imprimir "¿Cuántas materias llevas (max 9)?"	`printf("¿Cuántas materias llevas (max 9)?\n");`
leer mat	`scanf("%d",&mat);`
desde (i ←0, i < mat, i ←i +1)	`for (i=0;i<mat;i++)`
inicio	`{`
imprimir "Calificación de la",i+1," materia: "	`printf("Calificación de la %d materia: ",i+1);`
leer cal [i]	`scanf("%f",&cal[i]);`
prom ← prom + cal [i]	`prom=prom+cal[i];`
fin	`}`
cal[mat] ← prom / mat	`cal[mat]= prom/mat;`
imprimir "El promedio es ",cal[mat]	`printf("El promedio es %4.2f",cal[mat]);`
	`getch();`
	`return 0;`
fin	`}`

Ejemplos



Si *mat*=4, se guardarán en el arreglo cuatro calificaciones, *cal[0]=85, cal[1]=75, cal[2]=80, cal[3]=60,* la variable *prom* almacenará la suma que se va a ir acumulando en cada vuelta del ciclo *desde (for), prom=300*, luego *cal[4]=300/4*, para imprimir *El promedio es 75.00*, que se almacena en la variable *cal[4],* es decir en la quinta posición ya que son 4 materias.

Ejercicio 4. Almacene en un arreglo, *n* elementos de tipo entero, calcule el cuadrado de cada uno de los elementos y almacénelo en un segundo arreglo. Imprima ambos vectores.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
	`#include <math.h>`
principal ()	`main()`
inicio	`{`
entero x[10], y[10], i, n	`int x[10],i, n;`
	`float y[10];`
imprimir "Dame el tamaño del arreglo: "	`printf("Dame el tamaño del arreglo:");`
leer n	`scanf("%d",&n);`
desde (i←0,i<n, i←i+1)	`for (i=0; i<n; i++)`
inicio	`{`
imprimir "Dame un número: "	`printf("Dame un número: ");`
leer x[i]	`scanf("%d",&x[i]);`
y[i]←cuadrado x[i]	`y[i]= pow(x[i],2);`
fin	`}`
Imprimir " Los cuadrados son:"	`printf( "Los cuadrados son:\n");`
desde (i ← 0, i < n, i←i+1)	`for(i=0; i<n; i++)`
imprimir x[i], y[i]	`printf("%d,%.0f \n",x[i], y[i]);`
	`getch(); return 0;`
fin	`}`

En este ejercicio se declaran dos arreglos unidimensionales *x* y *y* de tamaño 10 y de tipo entero; *n* para el número de elementos que se desea almacenar e *i*, que sirve como subíndice del arreglo. Primero se solicita el número de elementos que se van a almacenar y se guarda en la variable *n*, luego con una estructura repetitiva *desde (for)* se inicializa *i* en cero y el último valor de *i* será *n*–*1*, ya que el arreglo, si es de ocho elementos, éstos estarán almacenados en *x[0]* hasta *x[7]*; dentro del ciclo *desde (for)* se van almacenando los valores en el arreglo *x* y luego se calcula el cuadrado de cada elemento y se almacena en el *arreglo y*. Cuando ya se hayan capturado los *n* elementos del arreglo *x*, de manera automática se habrán almacenado los *n* elementos del *arreglo y*. Para finalizar se imprimen los dos arreglos al mismo tiempo. Comenzando con el primer elemento del *arreglo x* y el primer elemento de *arreglo y*; después con el segundo y así sucesivamente hasta llegar al último elemento.

Ejemplos



Si *n=5, x[0]=7 almacenará en y[0]=49, x[1]=3 almacenará en y[1]=9, x[2]=8 almacenará en y[2]=64, x[3]=6 almacenará en y[3]=36, x[4]=2 almacenará en y[4]=4 e imprimirá 7, 49, 3, 9, 8, 64, 6, 36, 2, 4.*

Nota: En lenguaje C la *sintaxis* de la función *pow* es: pow(b,e) donde *b* es la base y *e* el exponente; dicha función regresa un valor real, por lo tanto el arreglo *y* tuvo que ser declarado float. Como los resultados de *y* son enteros cuando imprimimos su resultado utilizamos el formato %.0f (ya que no hay decimales).

Ejercicio 5. Calcule la desviación estándar *S* de cinco números.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
	`#include <math.h>`
principal ()	`main()`
inicio	`{`
entero num[5], i	`int num[5], i;`
real s←0,m←0	`float s=0.0,m=0.0;`
imprimir "introduce los 5 números"	`printf("introduce los 5 números \n");`
desde (i←0, i <5, i←i + 1)	`for (i=0;i<5;i++)`
inicio	`{`
leer num[i]	`scanf("%d", &num[i]);`
m ← m + num[i]	`m+=num[i];`
fin	`}`
m ←m / 5	`m= m/5;`
desde (i←0, i <5, i← i + 1)	`for (i=0;i<5;i++)`
s ← s + cuadrado(m-num[i])	`s+=pow((m-num[i]),2);`
s ← raizcuad (s / 5)	`s=sqrt(s/5);`
imprimir "La media es:", m	`printf("La media es: %.2f ",m);`
imprimir "La desviación estándar es",s	`printf("La desviación estándar es %4.2f",s );`
	`getch();`
	`return 0;`
fin	`}`

La fórmula a resolver es la siguiente:

$$S=\sqrt{\frac{\sum_{i=0}^{i=4}(m-num)^2}{5}}$$

Por lo tanto para calcular la desviación estándar primero debemos calcular la media y luego hacer el cuadrado de la diferencia de la media y de cada número para luego dividirla entre 5.

En este ejercicio *num* es la variable de tipo arreglo que almacena los cinco números. Las variables *s* y *m* se inicializan con cero porque son acumuladores y son de tipo real *(float)* porque calcularán fracciones. En el primer ciclo *desde (for)* se van almacenando los cinco valores del arreglo con ayuda de la instrucción *leer (scanf)*, en seguida se van acumulando los valores respectivos para cuando se salga del ciclo *desde (for)*. La media se calcula dividiendo los valores acumulados entre cinco. Luego entra a otro ciclo *desde (for)* en el cual se va haciendo la diferencia de la media *m* con cada número del arreglo *num* y el resultado lo eleva al cuadrado y lo va acumulando en la variable *s*. No debemos olvidar que la sumatoria Σ es el equivalente a acumular sumandos, por lo tanto acumularemos los valores de $(m\text{-}num_i)^2$ con el ciclo *desde (for)*, donde la *i* comienza con 0 hasta que llegue a 4, con incrementos de uno en uno. Posteriormente cuando salga del ciclo *desde (for)* calcula la desviación estándar dividiendo la variable *s* entre 5 y por último calcula la raíz cuadrada.

Ejemplos



Si los números son *x={8,6,9,5,7}* el resultado de la media *m=7* y la desviación estándar *s=1.41*.

Utilizamos la librería math.h ya que utilizamos la función matemática *pow*.

La salida en pantalla, después de ejecutar el programa sería:

introduce los 5 números

1 2 3 4 5

la media es: 3.00 y la desviación estándar es 1.41.

Ejercicio 6. Almacene en un vector, *n* elementos (máximo 30) e inserte un elemento *ni* en la posición *pos*, recorriendo los siguientes elementos una posición a la derecha.

Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
constante p←printf	#define p printf
constante s←scanf	#define s scanf
constante MAX←30	#define MAX 30
principal()	main()
inicio	{
entero lista[MAX],i,ni,n,pos	int lista[MAX],i,ni,n,pos;
imprimir "Dame el Tamaño del Arreglo "	p("Dame el Tamaño del Arreglo ");
leer n	s("%i",&n);
imprimir "Dame el No. a Insertar "	p ("Dame el No. a Insertar ");
leer ni	s("%i",&ni);
imprimir "Dame la Posición "	p("Dame la Posición ");
leer pos	s("%i",&pos);
desde (i←0,i<n,i←i+1)	for (i=0;i<n;i++)
inicio	{
imprimir "Dame el valor cuya posición es ",i	p ("Dame el valor cuya posición es %d ",i);
leer lista[i]	scanf("%d",&lista[i]);
fin	}
for (i←n,i>pos,i←i−1)	for (i=n;i>pos;i−−)
lista[i]←lista[i−1]	lista[i]=lista[i−1];
lista[pos]←ni	lista[pos]=ni;
desde (i←0,i<=n,i←i+1)	for (i=0;i<=n;i++)
imprimir lista[i]	p ("%d ",lista[i]);
	getch();
fin	}

En este ejercicio se declaran cinco variables de tipo entero: *lista* es un arreglo con un máximo de 30 elementos, *i* es un contador que enumera las posiciones del arreglo, *ni* es el número que se desea insertar, *n* es el tamaño del arreglo y *pos* es la posición donde se va a insertar el número. Se leen las variables *n*, *ni* y *pos*. Al conocer el valor de *n*, con un *desde (for)* se recorren las posiciones (0 a n−1), leyendo los valores respectivos del arreglo *lista*. Con otro *desde (for)* se va sustituyendo el valor de una casilla con el valor de la siguiente casilla, comenzando con la posición *n* hasta llegar a la posición deseada; por lo tanto el recorrido va de atrás hacia adelante. Al final se imprime la lista modificada, por lo tanto el tamaño del arreglo será n+1 y la condición del *desde (for)* será: *(i<=n). Si n=3, ni=10* y *pos=1*, en la instrucción lista[pos]← ni la posición 1 toma el valor de 10; el valor de la posición 0 se conserva: lista[0]=5; cabe recordar que las posiciones son las internas de la máquina ya que muchas veces el usuario visualiza desde la posición 1. Una posible corrida a mano será:

Primer desde (for)		***Segundo desde (for)***		***Tercer desde (for)***	
i	*lista [i]*	*i*	*lista [i]←lista [i-1]*	*i*	*lista [i]*
0	5	3	lista [3]←7	0	5
1	2	2	lista [2]←2	1	10
2	7	1	Se rompe	2	2
3	Se rompe			3	7
				4	Se rompe

Ejercicio 7. Almacene en un arreglo *a*, un conjunto *n* de elementos de tipo entero (max 15), almacene en el arreglo *b* los elementos del arreglo *a* de forma invertida.

Ejemplos 

Arreglo *a* = 2, 4, 6, 8, 10 almacena el usuario
Arreglo *b* = 10, 8, 6, 4, 2 lo realiza el programa

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
entero a[15],b[15],n, i,j	`int a[15],b[15],n, i,j;`
imprimir "Total de números"	`printf ( "Total de números");`
leer n	`scanf ("%d",&n );`
desde (i←0, i<n, i←i+1)	`for (i=0; i<n; i++)`
leer a[i]	`scanf ("%d",&a[i]);`
desde (i←0, j←n−1 ; i<n , j>=0 ; i← i + 1, j← j−1)	`for (i=0, j=n−1; i<n, j>=0; i++, j−−)`
b[i] ← a[j]	`b[i]=a[j];`
desde (i←0, i<n, i←i+1)	`for (i=0; i<n ; i++)`
imprimir a[i]	`printf ("%d \n",a[ i ] );`
desde (i←0, i<n, i←i+1)	`for (i=0; i<n ; i++)`
imprimir b[i]	`printf ("%d \n",b[ i ] );`
	`getch();`
	`return 0;`
fin	`}`

En este ejercicio se solicita el total de elementos y se almacena en la variable *n*, luego con la ayuda de un ciclo *desde* se llena el arreglo *a*, posteriormente, utilizando un ciclo *desde* con dos variables de control *i* para llevar la posición del arreglo *b* y *j* para llevar la posición del arreglo *a*, se van almacenando en la posición *b*[0], lo

que tiene el arreglo *a* [*n*-1], posteriormente se incrementa *i* y se decrementa *j*, para almacenar en la posición *b*[1], lo que tiene el arreglo *a* [*n*-2], así sucesivamente hasta almacenar en la posición *b*[*n*-1], lo que tiene el arreglo *a* [0].

Ejercicios complementarios de arreglos unidimensionales en pseudocódigo

Ejercicio 1. Complete el siguiente ejercicio que almacena en un arreglo unidimensional ocho números enteros y lo imprime de manera inversa.

```
principal ()
inicio
    ________________ x [ _________ ], i
    desde (i ← 0 , ________________, i← i+1)
    inicio
        imprimir " Introduce un numero"
        leer ________________
    fin
    imprimir " El arreglo de manera inversa es"
    desde (________________________, i← i−1)
        imprimir ___________________
fin
```

Ejercicio 2. Realice el pseudocódigo que sume dos vectores (max 10) elementos de tipo entero y el resultado lo almacene en un tercer vector. Imprima el vector resultante (c).

```
constante TAM ________________
principal ()
inicio
    real a[TAM], b[________], c[________]
    ________________ i,n
    imprimir "Número de elementos "
    leer n
    imprimir "Dame los valores del vector a"
    desde (i ←0, ________________, i ←i + 1)
    inicio
        imprimir " Introduce el valor de", i+1
        leer ________________
    fin
    imprimir "Dame los valores del vector b"
    ___________________ ( i ←0, i < n, i ←i + 1)
```

(*continúa*)

(*continuación*)

```
        inicio
                imprimir " Introduce el valor de", i + 1;
                leer ___________
        fin
        desde (i ←0, i < n, i ←i+1)
        inicio
                __________←a[i] + b[i]
                imprimir c[i]
        fin
fin
```

Ejercicio 3. Revise el siguiente pseudocódigo y conteste las siguientes preguntas.

```
principal ()
inicio
        real sal[30], salmay←0
        entero i , trab, sm
        imprimir "Total de trabajadores"
        leer   trab
        desde (i ← 0, i < trab, i ← i+1)
        inicio
                imprimir "dame el salario del trabajador" ,i+1
                leer sal [ i ]
                        si (sal [i ] > salmay)
                        inicio
                                sm ← i+1
                                salmay ← sal [i ]
                        fin
                fin
                imprimir    "El salario mayor es " , salmay," y pertenece al trabajador ", sm
fin
```

1. ¿Qué realiza el programa? ____

   ____

   ____

2. Total máximo de trabajadores que se pueden almacenar ______________________

   ____

3. ¿Qué almacena la variable *sm*? ______________________________________

   ____

4. Si hay dos salarios iguales ¿con cuál trabajador se queda la variable *sm*? ______________________________

____

5. ¿Qué almacena la variable *salmay*? ____

____

____

Ejercicio 4. Complete el siguiente ejercicio que almacena en un arreglo las temperaturas de las 24 horas del día (0-23) e imprima la temperatura más alta del día

```
principal ()
inicio
    real x [__________ ], i, may←0
    desde (i ← 0 , _____________, i← i+1)
    inicio
        imprimir " Introduce la temperatura"
        leer _______________
            si _________________
                may ← x [ i ]
    fin
    imprimir " La temperatura mayor fue",_______
fin
```

Ejercicio 5. Complete el siguiente pseudocódigo que realiza una búsqueda de un número real, si lo encuentra regresa la posición en la que se encontró y si no lo localizó, como posición se tiene el último valor de *i*.

```
principal ()
inicio
    real ___________________
    entero _________ , x
    desde (i ← 0, i < 20, i ← i+1)
    inicio
        imprimir "dame el número" ,i+1
        ______________ num [ i ]
    fin
    imprimir "Qué número deseas buscar"
    leer x
    i ← 0
    mientras (i < 20 y x <> num [i ])
        ________________
    imprimir "La posición del número es ", _______
fin
```

Ejercicios complementarios de arreglos unidimensionales en lenguaje C

Parte I. Complete los siguientes programas

Ejercicio 1. Leer una lista con un máximo de 30 elementos, buscar e imprimir la(s) posición(es) en la que se encuentra un valor *x*, dentro del arreglo. Si no se halla que imprima el letrero respectivo.

```
#include <conio.h>
#include <stdio.h>
#define MAX 30
main()
{
   int ____________[MAX],i,______,ban=0,x;
   printf("Dame el número de valores a ingresar \n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Dame el valor a ingresar %d",i+1);
      scanf("%d",&vector[i]);
   }
   printf("Dame el número a buscar ");
   ___________________
   for(i=0;i<n;i++)
      if(___________________)
      {
           printf("Lo encontré en la posición %d\n",i+1);
           ban=1;
      }
   if (______________) printf ("No lo encontré");
   getch();
   return 0;
}
```

Ejercicio 2. Leer una lista con un máximo de 30 elementos, crear dos vectores, el vector par contendrá los números pares del vector original y el non los impares. Al final imprima los tres vectores.

```
#include <conio.h>
#include <stdio.h>
#define MAX 30
main ()
{
     int lista[MAX],non[MAX],par[MAX],i,num,p=0,n=0;
     printf("dame un número de 1-30: ");
     scanf("%d",&num);
     for(i=0;i<num;i++)
     {
        printf("dame el número cuya posición o casillero es %d: ",i);
        scanf("%d", & lista[i]);
              if(lista[i]%2==0)
              {
                  par[p]=lista[i];
                  ____________
              }
              else
              {
                  n++;________________________
                      ________________________
              }
     }
     printf("lista original\n");
     for(i=0;i<num;i++)
        ___________________________________
     printf("\npares\n");
     for(i=0;i<p;i++)
        ___________________________________
     printf("\nnones\n");
     for(i=0;i<n;i++)
        printf("%d ",non[i]);
     getch();
     return 0;
}
```

Ejercicio 3. Imprimir la siguiente serie:

Lenguaje C

```
         8
       1 × 8+1 = 9
      12 × 8+2 = 98
     123 × 8+3 = 987
    1234 × 8+4 = 9876
   12345 × 8+5 = 98765
  123456 × 8+6 = 987654
 1234567 × 8+7 = 9876543
12345678 × 8+8 = 98765432
123456789 × 8+9 = 987654321
```

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
     int i, j, k, _____________, c;
     double ________;
     printf (“\t 8\n”);
     for (i=1;i<=9;i++)
     {
         ac=0;
         for (j=1;j<=9-i;j++)
             printf (“ “);
         for (c=0,j=1;j<=i;j++,c++)
         {
             printf (“%d”,j);
             num[c]=j;
         }
         for (c=0,j=i-1;j>=0;j--, c++)
             ac=ac+num[j]*pow(10,c);
         printf (“ X 8+%d=%.0f\n”,i,ac*8+i);
     }
     getch ();return 0;
}
```

Parte II. Realice la prueba de escritorio o corrida a mano del siguiente programa:

```c
#include <conio.h>
#include <stdio.h>
main()
{
   int ac=0, i,primero[15]={21,12,23,14,25,16,27,7,1,21,24,11};
   for (i=11;i>1;i=i-2)
   {
      ac=ac+primero[i-1]-i;
      printf(“%d,%d\n”,primero[i+2],ac);
   }
   getch(); return 0;
}
```

Parte III. Elabore la codificación en lenguaje C de los siguientes programas (realizando el cálculo e impresión respectiva):

1. Almacene en un arreglo las temperaturas de un día (0-23 horas). Calcule su media e imprimir la temperatura más alta y la más baja; asimismo imprimir la hora respectiva. Por ejemplo la temperatura media fue de 21.5 grados Celsius, la más alta de 29 grados Celsius a las 14 horas y la más baja de 9.4 grados Celsius a las 3 horas.
2. Almacene en un vector la temperatura de cada día de una determinada semana y que realice lo siguiente:
   a) La temperatura promedio.
   b) Un vector que contenga las diferencias de cada temperatura con respecto al promedio.
   c) La menor temperatura y el número de día en que ocurrió.
   d) La mayor temperatura y el número de día en que ocurrió.
3. Almacene en un arreglo, 20 elementos de tipo entero e imprimir:
   a) La suma de los elementos que ocupan posiciones pares.
   b) El mayor de los elementos que ocupan posiciones impares.
   c) La posición del mayor número par.
4. Almacene en un arreglo, *n* elementos (máximo 30) e imprimir la suma:
   a) de números pares.
   b) de números impares.
   c) total de los elementos del arreglo.
5. Almacene en una lista 15 números e imprimir cuántos son ceros, cuántos son negativos y cuántos positivos. Imprimir además la suma de los negativos y la suma de los positivos.
6. Inicialice un vector con 10 valores. Genere un vector *a* con los números negativos y otro vector *b* con los positivos o iguales a cero.
7. Almacene en un vector *n* elementos de tipo real (máximo 25) e imprima los siguientes valores:
   a) Máximo.
   b) Mínimo.
   c) La media de los que estén en posiciones pares.
   d) La varianza.
8. Genere aleatoriamente un vector de tamaño 20 con números entre 0 y 150 y genere otras tres listas con los siguientes criterios:
   a) Si los números están comprendidos entre 0 y 50 irán en la lista 1.
   b) Si los números están comprendidos entre 51 y 100 irán en la lista 2.
   c) Si los números son mayores a 101 irán en la lista 3.

   Al final imprimir las cuatro listas.
9. Almacene en un vector 10 números y cambie algún número por otro del mismo arreglo, realice el cambio y muestre el arreglo modificado.
10. Genere aleatoriamente un vector de tamaño 20 con números comprendidos entre 1 y 20 e imprimir:
    a) Suma de los elementos.
    b) Promedio de los elementos.
    c) Varianza y desviación estándar de los valores.
    d) Elemento que más se repite (moda).

11. Almacene en un vector de tamaño 10 números reales. Calcule el promedio e indique cuántos elementos del vector son mayores que el promedio y genere otro arreglo con los menores o iguales.
12. Almacene en un arreglo de *n* números enteros (máximo 30) y determine cuántos números terminan en dígito primo.
13. Convierta un número entero decimal a su equivalente en representación binaria.
14. Almacene en un arreglo *n* números enteros y determine ¿cuántas veces se repite cada uno de ellos?

Ejemplos

Si *n*=*6* y los elementos del arreglo son: 3,4,6,6,4,6, se imprimirá:

3=1

4=2

6=3

15. Guarde una cadena y genere otro vector con las veces que se repite cada letra.

Ejemplos



En la cadena CASAS generará el arreglo con los valores: 1,2,2,2,2.

16. Almacene en un vector *n* números (máximo 10), valide si el tamaño es par e invertir los elementos a la mitad del arreglo. Imprimir el vector original y el invertido.

Ejemplos



Si *n*=6, *v*=[1][2][3][4][5][6] *v*(invertido)=[3][2][1][6][5][4].

17. Almacene en dos vectores números reales *a* y *b* de 10 elementos cada uno, a partir de ellos genere un tercer vector con el siguiente criterio: sumar el primer elemento de *a* más el último elemento de *b* y luego el segundo elemento de *a* con el noveno elemento de *b* y así sucesivamente hasta llegar al décimo elemento de *a* más el primer elemento de *b*. Imprimir las sumas almacenadas en el vector *c*.
18. Genere aleatoriamente un arreglo unidimensional de *n* enteros comprendidos entre 1 y 100 e imprima otro vector pero sin los elementos repetidos.
19. Almacene un vector de longitud *n* ordenado en forma ascendente y un elemento *x* del mismo tipo que los elementos del vector, intercalar *x* en el vector *v* de manera que siga ordenado.
20. Almacene 2 vectores *a* y *b* de longitudes *n* y *m*, el primero ordenado en forma ascendente y el segundo ordenado de manera descendente, crear un nuevo vector *c* de *n* + *m* elementos intercalando los elementos de *a* y *b* de modo que *c* quede ordenado en forma ascendente.
21. Imprimir las siguientes series:

1 × 9+2 =11	1
12 × 9+3 =111	1 × 1=1
123 × 9+4 =1111	11 × 11=121
1234 × 9+5 =11111	111 × 111=12321
12345 × 9+6 =111111	1111 × 1111=1234321
123456 × 9+7 =1111111	11111 × 11111=123454321
1234567 × 9+8 =11111111	111111 × 111111=12345654321
12345678 × 9+9 =111111111	1111111 × 1111111=1234567654321
123456789 × 9+10=1111111111	11111111 × 11111111=123456787654321
	111111111 × 111111111=12345678987654321

22. Leer un número entero positivo e imprimirlo separado dígito por dígito.

Ejemplos



Si *N*=123456, imprimirá 1 2 3 4 5 6.

4.3 Arreglos bidimensionales (matrices o tablas)

Un arreglo bidimensional es un conjunto de *n* elementos del mismo tipo almacenados en memoria contigua en una matriz o tabla. A diferencia de los arreglos unidimensionales que sólo requieren de un subíndice, los arreglos bidimensionales para acceder a cada elemento del arreglo requieren de dos índices o subíndices declarados en dos pares de corchetes, donde el primer corchete se refiere al tamaño de filas y el segundo al tamaño de columnas.

Ejemplos



Para declarar una tabla de 3 filas y 5 columnas, lo haremos de la siguiente forma:

Pseudocódigo	Lenguaje C
tipo_dato ident_arr [tam_ fila][tam_col]	`tipo_dato ident_arr [tam_fila][tam_col];`
entero b[3][5]	`int b[3][5];`

donde:

tipo_dato es el tipo de dato de todo el arreglo.

ident_arr es el nombre del arreglo.

tam_fila es el total de filas.

tam_col es el total de columnas.

Nota: El número de elementos de una matriz será tam_fila × tam_col. En el ejemplo anterior son 3 × 5 = 15 celdas de tipo entero. Los elementos del arreglo *b* están ordenados de la manera siguiente:

b	Col 0	Col 1	Col 2	Col 3	Col 4
Fila 0	b[0][0]	b[0][1]	b[0][2]	b[0][3]	b[0][4]
Fila 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]
Fila 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]	b[2][4]

4.3.1 Inicialización de arreglos bidimensionales

En el momento de declarar el arreglo, se especifican los valores.

Sintaxis:

Pseudocódigo	Lenguaje C
t_dato identif [fil][col] ← {valores}	`t_dato identif[fil][col]={valores};`
entero a[3][3]←{1,2,3,4,5,6,7,8,9}	`int a[3][3]= {1,2,3,4,5,6,7,8,9};`

a	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

4.3.2 Lectura e impresión de unos arreglos bidimensionales

Para la lectura la computadora requiere de dos ciclos anidados (para ubicar la fila y la columna), y la instrucción *leer (scanf)* o *leercad (gets)*, almacenando con ello los valores en cada celda de la tabla o matriz.

El siguiente segmento de programa muestra cómo se pueden almacenar datos en una matriz *mat* de 3 filas y 4 columnas, se utiliza la instrucción *leer (scanf)* para guardar o leer los datos:

Pseudocódigo	Lenguaje C	*Qué realiza*
desde (i←0, i < 3, i← i+1)	`for(i=0;i<3;i++)`	Recorre cada fila (i),
desde (j←0, j< 4, j← j+1)	`for(j=0;j<4;j++)`	Recorre cada columna (j)
leer mat[i][j]	`scanf("%d",&mat[i][j])`	Almacena en la fila i y columna j correspondiente

Nota: Cuando existen dos ciclos anidados primero se inicializa el ciclo de fuera, posteriormente realiza el que se encuentra dentro y hasta que termine con él regresa al ciclo de fuera; en este caso, primero inicializa el ciclo *i* y hasta que termina con todas las columnas (j) de la primera fila (0) y continúa con la segunda fila (1), iniciando otra vez con la columna (j=0).

El siguiente segmento de programa muestra cómo se pueden imprimir los datos almacenados en una matriz *mat* de 3 filas y 4 columnas. Se utiliza la instrucción imprimir *(printf)* para escribir o mostrar el resultado:

Pseudocódigo	Lenguaje C	*Qué realiza*
desde (i←0, i < 3, i← i+1)	`for(i=0;i<3;i++)`	Recorre cada fila (i),
desde (j←0, j< 4, j← j+1)	`for(j=0;j<4;j++)`	Recorre cada columna (j)
imprimir mat[i][j]	`printf("%d", mat[i][j])`	Imprime el valor almacenado en la fila *i* y columna *j* correspondiente

4.3.3 Modificación de un elemento de una matriz

Los elementos de una matriz se pueden modificar en cualquier momento, sólo es necesario especificar el nombre del arreglo bidimensional, la posición tanto de la fila como de la columna y el nuevo valor. En seguida se muestra la sintaxis a seguir:

Pseudocódigo	Lenguaje C
tipo_dato ident_arr [fil][col] ← valor	`tipo_dato ident_arr [fil][col] =valor;`
entero b[3] [5] ← 18	`int b[3][5]=18;`

Donde *valor* es un dato o el resultado de una llamada a función o de alguna operación lógica o aritmética, etc. En este ejemplo se le asigna directamente el valor *18* a la matriz *b* en la *cuarta fila* y *sexta columna*.

Muchas manipulaciones con arreglos utilizan la estructura de control repetitiva *desde (for)*. Por ejemplo, el siguiente ejemplo define todos los elementos en el tercer renglón o fila del arreglo *a* (la fila 2 almacena los elementos de la tercera fila); el único subíndice que se modifica es *j*, que sirve para cambiar el subíndice de las columnas:

Pseudocódigo	Lenguaje C
entero a[4][4]	`int a[4][4];`
desde (j ← 0, j<4, j ←j +1)	`for (j=0; j < 4; j+ +)`
a[2][j] ←5	`a[2][j]=5;`

El ciclo *desde (for)* anterior es equivalente a los enunciados de asignación siguientes: a[2][0] = 5, a[2][1] = 5, a[2][2] = 5, a[2][3] = 5; es decir, se le asigna a toda la tercera fila, el valor de 5.

4.3.4 Ejercicios resueltos de arreglos bidimensionales

El ejercicio 1 incorpora el concepto de cadena de caracteres; nótese que el *arreglo x* que se declara es de tipo caracter ya que en cada celda o posición almacenará un carácter y para ello se requieren dos valores: *[50]* que se refiere al número de alumnos, y *[20]* para almacenar el nombre de cada uno de los alumnos (recuerde que \0 ocupa un lugar) este tipo de arreglo se denomina bidimensional. Se solicita el total de alumnos indicando el máximo de 50, y se almacena en la variable *alum*; dentro del ciclo *desde (for)* se va solicitando el nombre de cada alumno, para esto utilizamos una función de cadenas del lenguaje C, *leercad (gets)* equivalente al *leer* ya utilizada, solamente que éste es especial para cadenas (cuando se tienen problemas con el formato %s, es conveniente cambiar scanf("%s",&x[i]); por gets(x[i])) además la función *gets()* requiere de la función *fflush* (vea anexo A).

Ejercicio 1. Guardar los nombres de los alumnos de un salón de clases en un arreglo y posteriormente imprimirlos.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main()`
inicio	`{`
caracter x[50][20]	`char x[50][20];`
entero alum, i	`int alum,i;`
imprimir "Cuántos alumnos hay (max 50)"	`printf("Cuántos alumnos hay (max 50)");`
leer alum	`scanf("%d",&alum);`
desde (i←0, i <alum, i←i+1)	`for (i=0;i< alum;i++)`
inicio	`{`
imprimir "Nombre del alumno ",i+1	`printf("Nombre del alumno %d",i+1);`
leercad x[i]	`fflush (stdin); gets(x[i]);`
fin	`}`
desde (i←0, i<alum, i← i+1)	`for (i=0;i<alum;i++)`
imprimir " alumno ",i+1," nombre : ",x[i]	`printf ("alumno %d nombre: %s\n",i+1,x[i]);`
	`getch(); return 0;`
fin	`}`

Note que el arreglo *x* sólo tiene un índice *i*, esto quiere decir que se está almacenando en uno de los 50 posibles lugares y en los 20 posibles sitios que tiene para cada carácter.

El segundo *desde (for)* sirve para imprimir en pantalla la lista de nombres.

Ejercicio 2. Almacenar en una matriz de $n \times m$ (máximo 10×10) números reales. Imprimir los elementos de la diagonal principal de la matriz e imprimir la suma dichos elementos.

Pseudocódigo
principal ()
inicio
real a[10][10], acum←0
entero i,j,n,m
imprimir "Número de filas "
leer n
imprimir "Número de columnas "
leer m
desde (i←0, i < n, i ← i+1)
inicio

(*continúa*)

(*continuación*)

Pseudocódigo

```
            imprimir "Lectura de la fila: " ,i+1, "de la matriz a :"
            desde (j←0, j<m, j←j+1)
            inicio
                  imprimir "a",i+1,j+1
                  leer a[i][j]
            fin
      fin
      desde (i←0, i<n, i←i+1)
            acum←acum+a[i][i]
      imprimir ("Acumulación de la diagonal principal es: ",acum
fin
```

En este ejercicio se incorporan los arreglos bidimensionales de tipo numérico; almacenaremos una matriz *a*, de *[10][10]*, reservando en memoria *100* espacios de tipo real, aunque el usuario puede elegir cuántas filas *n* y cuántas columnas *m*; al igual que en los arreglos unidimensionales, la manera más sencilla de llenar un arreglo es usando un ciclo *desde (for)*; para llenar una matriz o tabla se tendrán que utilizar dos *ciclos desde (for) anidados*, el primero para las filas y el segundo para las columnas, es decir cuando la variable *i = 0* entra al segundo *ciclo desde (for) j= 0* y almacenará el valor en a*[0][0]*, luego *j= 1* y almacenará el valor *a[0][1]* y así sucesivamente hasta llegar a la *m* columna *a[0][m-1]*. Al terminarse el segundo *ciclo desde (for)* se regresa al primer *ciclo desde (for)* y se incrementa *i* en uno para tomar *i = 1* y entra otra vez al segundo *ciclo desde (for)*, iniciando el valor de *j = 0*, para almacenar el valor *a[1][0]*, luego en *a[1][1]*, hasta llegar a la última columna *m*. Así ocurre sucesivamente hasta llegar a la última fila y por consiguiente a la última columna. Una vez llena la matriz *a*, se irán acumulando los valores de la diagonal principal en la variable *acum*, para esto sólo se necesita un *ciclo desde (for)*, ya que con un solo índice podemos ubicarnos en *a[0][0], a[1][1]*, hasta *a [n-1] [n-1]*, para al final imprimir el valor de la variable *acum*.

Ejemplos



Si la matriz es de *4 × 2*, es decir *n= 4* y *m= 2*, en esta matriz se llenó primero la fila 0, luego la fila 1 y así sucesivamente hasta la fila 3, y por cada fila sus dos columnas, es decir se fueron llenando a[0][0]=7, a[0][1]=4, a[1][0]=3, a[1][1]=9, a[2][0]=1, a[2][1]=8, a[3][0]=5, a[3][1]=2, para luego acumular solamente la diagonal principal *a[0][0]=7, a[1][1]=9* y al final la suma de esta diagonal *acum = 16*.

a	*0*	1
0	7	4
1	3	9
2	1	8
3	5	2

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
main()
{
     float a[10][10], acum=0;
     int i,j,k,n,m;
     printf("Número de filas ");
     scanf("%d",&n);
     printf("Número de columnas ");
     scanf("%d",&m);
     for (i= 0;i< n;i++)
     {
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Número de filas 2
Número de columnas 2
Lectura de la fila 1 de la matriz a:
a(1, 1)=5
a(1, 2)=2
Lectura de fila 2 de la matriz a:
a(2,1)=3
a(2,2)=7
Acumulación de la diagonal principal es 12.00

(*continúa*)

(*continuación*)

Lenguaje C

```
         printf("Lectura de la fila %d de la matriz a :\n",i+1);
         for (j=0; j<m; j++)
         {
             printf("a(%d,%d)=",i+1,j+1); scanf("%f",&a[i][j]);
         }
     }
     for (i = 0; i< n;i++)
         acum=acum+a[i][i];
     printf("Acumulación de la diagonal principal es: %6.2f",acum);
     getch(); return 0;
}
```

Ejercicio 3. Realice el programa para almacenar en una matriz de $f \times c$ (máximo 20×20) un conjunto de elementos de tipo real y almacenar en una segunda matriz la raíz cuadrada de cada elemento. Imprimir la matriz original y posteriormente la matriz que almacenó las raíces cuadradas.

Pseudocódigo

```
principal ()
inicio
  real x[20][20], y[20][20]; entero i, j, f, c
  imprimir "Número de filas "; leer f
  imprimir "Número de columnas "; leer c
  imprimir "Dame los valores de la matriz a: "
  desde (i ← 0, i < f, i ←i + 1)
     desde (j ← 0, j < c, j ← j + 1)
     inicio
        imprimir "a(",i," ,",j,")="
        leer x[i][j]
        y[i][j] ← raizcuad(x[i][j])
     fin
  imprimir "Matriz Original"
  desde (i ← 0, i <  f ,i ←i +1)
  inicio
     desde ( j ← 0, j < c, j ← j +1)
        imprimir x[i][j]
     salto_linea
  fin
  imprimir " Matriz Raiz cuadrada"
   desde (i ← 0, i <  f, i ← i + 1)
   inicio
      desde ( j ← 0, j < c, j ← j + 1)
          imprimir y[i][j]
      salto_linea
   fin
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
   float x[20][20], y[20][20]; int i,j,f,c;
   printf("Número de filas "); scanf("%d",&f);
   printf("Número de columnas "); scanf("%d",&c);
   printf("Dame los valores de la matriz a:\n ");
   for (i=0;i< f;i++)
      for (j=0;j<c;j++)
      {
         printf("a(%d,%d)= ",i,j);
         scanf("%f",&x[i][j]);
         y[i][j]=sqrt(x[i][j]);
      }
   printf("Matriz Original\n");
   for (i=0;i<f;i++)
   {
      for (j=0;j<c;j++)
         printf ("%5.2f ", x[i][j]);
      printf ("\n");
   }
   printf("Matriz Raíz cuadrada\n");
   for (i=0;i< f;i++)
   {
      for (j=0;j<c;j++)
         printf ("%5.2f ", y[i][j]);
      printf ("\n");
   }
   getch(); return 0;
}
```

El ejercicio 3 tiene similitud con el ejercicio resuelto 4 de arreglos unidimensionales, la diferencia es que en éste se almacenan los valores en la matriz *x*; además en lugar de manejar el cuadrado del vector *x*, se calcula la raíz cuadrada de cada elemento de la matriz *x*; y por último se almacena en la misma posición de la matriz *y*.

Si *f* = 3 y *c* = 2:

x	0	1
0	7.5	5.4
1	3.6	7.9
2	1.8	10.5

y	0	1
0	2.74	2.32
1	1.90	2.81
2	1.34	3.24

Cuando se almacena en *x* [0][0] = 7.5, se realiza el cálculo y se almacena en *y* [0][0] = 2.74. Así sucesivamente hasta terminar con el elemento en la posición *x* [2][1] = 10.5 almacenando en *y* [2][1] =3.24. Se imprime primero la ***matriz x, y*** posteriormente la ***matriz y***.

Ejercicio 4. Realice el programa para sumar dos matrices y almacenar el resultado en una tercera matriz. Imprimir al final el resultado.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
principal ()	`main ()`
inicio	`{`
real a[10][10], b[10][10], c[10][10]	`float a[10][10], b[10][10], c[10][10];`
entero i, j, n, m	`int i,j,n,m;`
imprimir "Número de filas "; leer n	`printf( "Número de filas "); scanf("%d",&n);`
imprimir "Número de columnas "; leer m	`printf( "Número de columnas "); scanf("%d",&m);`
imprimir "Dame los valores de la matriz a"	`printf( "Dame los valores de la matriz a\n");`
desde (i ← 0, i < n, i ← i + 1)	`for (i =0; i <n; i++)`
desde (j ← 0, j< m, j ← j + 1)	`for (j =0; j <m; j++)`
inicio	`{`
imprimir "a(",i," ,",j,") ="	`printf("a(%d,%d)= ",i,j);`
leer a[i][j]	`scanf("%f",&a[i][j]);`
fin	`}`
imprimir "Dame los valores de la matriz b"	`printf("Dame los valores de la matriz b\n");`
desde (i ← 0, i < n, i ← i +1)	`for (i = 0; i < n; i++)`
desde (j ← 0, j < m, j ← j +1)	`for ( j =0; j< m; j++)`
inicio	`{`
imprimir "b(",i,", ",j,")="	`printf("b(%d,%d)= ",i,j);`
leer b[i][j]	`scanf("%f",&b[i][j]);`
fin	`}`
desde (i ← 0, i < n, i ← i +1)	`for (i =0; i < n; i++)`
desde (j ← 0, j < m, j ← j +1)	`for (j= 0; j < m; j++)`
c[i][j]=a[i][j]+b[i][j]	`c[i][j]=a[i][j]+b[i][j];`
imprimir "Matriz Suma:"	`printf("Matriz Suma:\n");`
desde (i ← 0, i < n, i ← i +1)	`for (i =0; i < n; i++)`
desde (j ← 0, j <m, j ← j +1)	`for (j=0; j <m; j++)`
imprimir c[i][j]	`printf("%f\n",c[i][j]); getch(); return 0;`
fin	`}`

En este ejercicio 4 necesitamos primero almacenar los valores de las matrices *a*, y *b*, para luego hacer la suma y almacenarla en la matriz *c*; se declaran tres matrices *a*, *b* y *c*, del mismo tamaño. Luego se pregunta el total de filas y se guarda en *n* y el total de columnas y se guarda en *m*. Se capturaron los datos de las dos matrices por separado, pero se pudieron haber pedido en el primer ciclo anidado donde se almacenaron los datos de la matriz *a*. Por lo tanto, encontramos *dos ciclos desde (for) anidados* para guardar los valores de la *matriz a*. Debemos recordar que internamente se inicia con el elemento cuyas coordenadas son [0][0] aunque se visualiza como [1][1], para aprovechar al máximo la memoria. Tenemos otros *dos ciclos desde (for) anidados* para guardar los valores de la *matriz b*, y los últimos *dos ciclos desde (for) anidados* para sumar los valores de *a* y *b* y guardar los valores de la *matriz c*; nótese que para almacenar los valores de la matriz c[i][j] se toman los valores de a[i][j] y b[i][j], es decir si se hace la suma el valor de c[0][0] = a[0][0] + b[0][0], c[0][1] = a[0][1]+ b[0][1] y así sucesivamente. Los dos últimos *ciclos desde (for) anidados* sirven para imprimir el resultado de la *matriz c*.

Ejemplos



Si la matriz es de 3 × 2, es decir *n* = 3 y *m* = 2, en la matriz *a* se llenó primero la fila 0, luego la fila 1 y así sucesivamente hasta la fila 2, y por cada fila sus dos columnas, es decir se fueron llenando a[0][0]=8, a[0][1]=4, a[1][0]=2, a[1][1]=7, a[2][0]=5, a[2][1]=3; luego se llena la *matriz b*, b[0][0]=7, b[0][1]=6, b[1][0]=3, b[1][1]=9, b[2][0]=1, b[2][1]=8, para realizar la suma y almacenarlos en la *matriz c*, como se muestra c[0][0]= 8 + 7 =15, y así sucesivamente para que al final se impriman los valores de la *matriz c*. Veamos las matrices:

a	0	1
0	8	4
1	2	7
2	5	3

b	0	1
0	7	6
1	3	9
2	1	8

c	0	1
0	15	10
1	5	16
2	6	11

Ejercicio 5 Almacene en una matriz las calificaciones de los exámenes (máximo 10) presentados durante un curso por un grupo de estudiantes (máximo 50); calcular el promedio individual, imprimir en pantalla el número de cada alumno, sus calificaciones y el promedio del grupo.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
constante MAX_ALUM ← 50	`#define MAX_ALUM 50`
constante MAX_MAT ← 10	`#define MAX_MAT 10`
principal ()	`main()`
inicio	`{`
real calif [MAX_ALUM][MAX_MAT], acum	`float calif[MAX_ALUM][MAX_MAT],acum;`
entero i, j, t_alum, t_exa	`int i,j,t_alum,t_exa;`
imprimir "Dame Total de Alumnos";	`printf("Dame Total de Alumnos")`
leer t_alum	`scanf("%d",&t_alum);`
imprimir "Dame Total de Exámenes";	`printf("Dame Total de Exámenes")`
leer t_exa	`scanf ("%d",&t_exa);`
desde (i ← 0, i < t_alum, i ← i+1)	`for (i=0;i<t_alum;i++)`
inicio	`{`
acum ← 0	`acum=0;`
desde (j ← 0, j < t_exa, j ← j+1)	`for (j=0;j<t_exa;j++)`
inicio	`{`
imprimir "Dame el Examen", j+1,"del alumno: ",i+1	`printf("Dame el Examen %d del alumno %d:", j+1,i+1);`
leer calif[i][j]	`scanf("%f",&calif[i][j]);`
acum ← acum+calif[i][j]	`acum=acum+calif[i][j];`
fin	`}`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
calif[i][j] ← acum / t_exa	`calif[i][j]=acum/t_exa;`
fin	`}`
imprimir "No. Alumno"	`printf("No. Alumno \n");`
desde (i ← 0, i < t_alum, i ← i+1)	`for (i=0;i<t_alum;i++)`
inicio	`{`
imprimir i+1	`printf("%9d",i+1);`
desde (j ← 0, j<=t_exa, j ← j+1)	`for (j=0;j<=t_exa;j++)`
imprimir calif[i][j]	`printf("%7.2f", calif [i][j]);`
salto_linea	`printf("\n");`
fin	`}`
	`getch(); return 0;`
fin	`}`

En este ejercicio se trata de almacenar en una *matriz calif* todas las calificaciones y en la última columna que almacene su promedio. Utilizamos una constante para definir el tamaño de la matriz, considerando cuando máximo 50 alumnos y 10 materias. Se pregunta el total de alumnos y se almacena *t_alum*, y luego el total de exámenes, almacenándose en *t_exa*. El ciclo exterior *i (renglones)* controla los alumnos y el interior *j (columnas)* los exámenes; antes de entrar al *ciclo j* se inicializa *acum=0*, esto es necesario ya que cada vez que se solicite las calificaciones de cada alumno su contador o acumulador inicie en cero. Luego entramos al segundo *ciclo desde (for)*, que sirve para solicitar las calificaciones de cada materia y poder almacenarlas en la posición según sea el número de alumno y según la calificación del examen. Todas las calificaciones del alumno uno se almacenarán en la fila 0 y en la respectiva columna hasta la columna *t_exa-1*. Al mismo tiempo estas calificaciones se van acumulando en la variable *acum*; cuando se sale del *ciclo desde (for) j* se hace la operación para calcular el promedio de cada alumno y lo almacena en la última columna de cada alumno; es decir, si es el alumno uno y llevó cuatro materias, éstas estarían almacenadas en *calif[0][0]=85, calif[0][1]=75, calif[0][2]=90, calif[0][3]=80*, así que su acumulado será *acum = 330* para luego dividirlo entre 4 para calcular el promedio y almacenarlo en la posición *calif[0][4]=82.5*. Luego encontramos el fin del primer ciclo, *desde (for)* el cual incrementa la variable *i* en uno para pasarse al siguiente alumno, y la variable *acum* se inicializa otra vez en cero *(acum = 0)* para iniciar con los exámenes del segundo alumno y así sucesivamente. La matriz leída crece de tamaño *t_alum x t_exa a t_alum x t_exa+1*, ya que el promedio ocupa una columna más. Los dos últimos *desde (for) anidados* sirven para mostrar toda la matriz, incluyendo el promedio calculado de cada alumno. Veamos un ejemplo de la tabla respectiva:

calif	0	1	2	3	4
0	85	75	90	80	82.5
1	70	80	70	80	75
2	95	90	95	90	92.5
3	60	70	60	70	65
4	80	90	90	80	85

Ejercicio 6. Encuentre el elemento mayor y el menor en una matriz de tamaño (10 × 10).

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
constante MAX ← 10	`#define MAX 10`
principal ()	`main()`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
inicio	{
entero a[MAX][MAX],i,j,n,m,may,min	int a[MAX][MAX],i,j,n,m, may,min;
imprimir "Dame el total de filas:";	printf("Dame el total de filas:");
leer m	scanf ("%d",&m);
imprimir "Dame el total de columnas:"	printf("Dame el total de columnas:");
leer n	scanf("%d",&n);
desde (i ← 0,i<m,i ← i+1)	for(i=0;i< m;i++)
desde (j ← 0,j < n,j ← j+1)	for(j=0;j<n;j++)
inicio	{
imprimir "a(",i,",",j,")="	printf("a(%d,%d)=",i,j);
leer a[i][j]	scanf("%d",&a[i][j]);
fin	}
may ← a[0][0]	may=a[0][0];
min ← a[0][0]	min=a[0][0];
desde (i ← 0,i < m, i ← i+1)	for(i=0;i<m;i++)
desde (j ← 0,j < n, j ← j+1)	for(j=0;j<n;j++)
inicio	{
si (a[i][j] > may) may ← a[i][j]	if(a[i][j] >may) may =a[i][j];
sino si (a[i][j] < min)	else if(a[i][j]<min)
min ← a[i][j]	min=a[i][j];
fin	}
imprimir "Número mayor ",may	printf("Número mayor %d\n", may);
imprimir "Número menor ",min	printf("Número menor %d",min);
	getch(); return 0;
fin	}

En este ejercicio se declara una matriz de *m* (filas) y *n* (columnas); posteriormente se llena utilizando dos ciclos anidados del *desde (for)*, la variable *i* sirve como subíndice para las filas y la variable *j* para las columnas. A las variables *may* y *min* se les asigna el valor del elemento de la matriz que se encuentra en la primera posición, es decir a[0][0], y otra vez se utilizan dos ciclos anidados del *desde (for)*, ahora para ir comparando cada elemento de la matriz con el elemento mayor almacenado en may y con el elemento menor almacenado en min. Para poder encontrar el mayor y el menor se utiliza una estructura de control selectiva doble anidada, para ir revisando si el elemento es mayor: *si(a[i][j]>may)* entonces se almacena este nuevo valor en la variable may. Si fue falso se revisa si es menor con la condición *si(a[i][j]<min)*; si se cumple se almacena este nuevo valor en la variable min.

Si se llena la matriz *a* con cinco filas y cuatro columnas; si se almacenan los valores del ejemplo anterior en ella. Después may = 85 y min = 85, se va revisando cada elemento de matriz, la condición (75 > 85) es falsa y queda may = 85, luego compara la condición (75<85) es verdadera y se almacena en min = 75 y así sucesivamente hasta llegar al último elemento almacenado en a[4][3]. Dicho programa imprimiría Número mayor 95 y menor 60.

Ejercicios complementarios de arreglos bidimensionales en pseudocódigo



Ejercicio 1. Guarde los nombres de los estados y el distrito de la República Mexicana en un arreglo y posteriormente imprimirlos.

```
principal ()
inicio
    caracter est[________][20]
    entero i
    desde (i ← 0, i < ________ , i ← i+1)
    inicio
        imprimir "Dame el nombre del estado ",i+1
        leercad ________________________
    fin
    desde (i ← 0, i< ________________________, i ← i+1)
        imprimir " estado ",i+1," es : ",________________________
fin
```

Ejercicio 2. Complete el siguiente pseudocódigo para llenar una matriz cuadrada (máximo 15 × 15), e imprimir la diagonal principal.

```
constante MAX ________
principal ()
inicio
    entero x[________][________],i,j,n
    imprimir "Dame el total de filas o columnas"
    leer ____________
    desde (i←0, ____________, i← i+1)
        desde (j←0, ______________, j ←j+1)
                      ______________
    desde (i←0, ______________, i ←i+1)
        imprimir _________________
fin
```

Ejercicio 3. Almacene en una matriz de 10 ×10 (máx), un conjunto de números enteros positivos y encuentre el elemento mayor de la matriz y decir en qué posición lo encontró.

```
principal()
inicio
    entero m[10][10], i, j, f, c, pf, pc, mayor←0
    leer f,c
    desde (____________, i < f, i ← i+1)
        desde (j←0, j< ____________, j ←j+1)
```

(continúa)

(*continuación*)

```
            inicio
                  ____________________
                  si(m[i][j]> __________)
                  inicio
                        mayor ← ____________
                        pf ← i
                        ____________
                  fin
            fin
      imprimir "el numero mayor es",____________" y se encontró en la fila",____________, " y columna",____________
fin
```

Ejercicio 4. Complete el siguiente pseudocódigo que almacena en una matriz de 4 × 4 números reales y realiza la suma de toda la fila y el resultado lo almacena en la misma fila y última columna disponible, y la suma de toda la columna y el resultado lo almacena en la misma columna y última fila disponible. Imprime solamente la fila de las sumas y la columna de las sumas.

```
principal ()
inicio
      real num [_____][_____], acum; entero i, j
      imprimir "Almacena los números y suma filas"
      desde (i ← 0, ____________________, i ← i+1)
      inicio
            acum ← 0
            desde (j ← 0, ____________________, j ← j+1)
            inicio
                  imprimir "Introduce el numero ",i+1, j+1
                  leer ____________________
                  acum ← acum + num[i][j]
            fin
            num[i][j] ← ____________________
      fin
      imprimir "Sumar las columnas"
      desde (j ← 0, ____________________, j ← j+1)
      inicio
            acum ←0
            desde ________________________________
                  acum ←acum + num[i][j]
            ____________________ ←acum
      fin
      imprimir "La Suma de las filas"
      desde (i ← 0, ____________________, i ←i+1)
            imprimir ____________________
      imprimir "La Suma de las columnas"
      desde (j ←0, ____________________, j ← j+1)
            imprimir ____________________
fin
```

Ejercicios complementarios de arreglos bidimensionales en lenguaje C

Parte I. Complete el siguiente programa, el cual calcula e imprime la multiplicación de dos matrices cuadradas

```
#include <stdio.h>
#include <conio.h>
main()
{
   int i,j,k,n,ac=0;
   int a[12][12],b[12][12],c____________;
   printf("Dame el tamaño de la matriz ");
   __________________
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
         printf("Dame el elemento de a con coordenadas %d,%d ",i+1,j+1);
         __________________________
         printf("Dame el elemento de b con coordenadas %d,%d ",i+1,j+1);
         scanf("%d",________________);
      }
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
         ac=0;
         for(k=0;k<n;k++)
             ac=ac+a[i][k]*b[k][j];
         c[i][j]=______;
      }
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
         printf("%d ",_______________);
      printf("\n");
   }
   return 0;
   getch();
}
```

Parte II. Realice la prueba de escritorio o corrida a mano de los siguientes programas:

Ejercicio 1

```
main()
{
  int i,j,segundo[15];
  int primero[15]={1,12,3,14,5,26,7,5,3,1};
```

Ejercicio 2

```
main()
{
  int i,j;
  int tercero[10][10];
```

(continúa)

(*continuación*)

```
   for (i=2;i<7;i++)
      segundo[i]=primero[2*i-2]+i;
      for (j=2;j<5;j++)
      {
         printf("%d, ",primero[j+1]);
         printf("%d\n", segundo[j+2]);
      }
   getch();
   return 0;
   }
```

```
   for(i=0;i<5;i++)
      for(j=0;j<5;j++)
         tercero[i+1][j]=-j/2+i*3+3;
   for(i=1;i<5;i+=2)
   {
      j=2;
      while (j<5)
      {
         printf("%d,%d,%d\n",i,j, tercero [i][j]);
         j=j+2;
      }
   return 0;
   } getch();
}
```

Ejercicio 3

```
main()
{
   int i,j,vec[5]={3,1,6,4,5},mat[3][4];
   for(i=0;i<3;i++)
     for(j=0;j<4;j++)
        mat[i][j]=vec[i]+vec[j]+j;
   for(i=0;i<2;i++)
   {
     j=1;
     while (j<5)
     {
        printf("%d\n",mat[i+1][j-i]);
        j+=2;
     }
   } getch(); return 0;
}
```

Ejercicio 4

```
main()
{
   int a[3][3]={11,12,31,4,5,16}, c[3][3];
   int i,j,b[3]={7,8,9};
   for(i=0;i<3;i++)
     for(j=0;j<3;j++)
        c[i][j]=a[j][i]+b[j]+j;
   for(i=1;i<2;i++)
   {
     j=0;
     while(j<3)
     {
        printf("%d,%d\n",i*b[j], c[i][j]);
        j+=2;
     }
   } getch(); return 0;
}
```

Parte III. Elabore la codificación en lenguaje C de los siguientes programas (realizando el cálculo e impresión respectiva):

1. Almacene los elementos de una matriz m×n (máximo 10×10) e imprimir cuántos números son ceros, cuántos son negativos, y cuántos positivos. Imprimir además la suma de los negativos y la suma de los positivos.
2. Leer los elementos de una matriz m×n (máximo 10×10) y multiplicarla por un escalar.
3. Almacene los elementos de una matriz m×n (máximo 10×10) e imprimir :

   a) Suma de los elementos.

   b) Promedio de los elementos.

   c) Elemento que más se repite (moda).

4. Almacene los elementos de una matriz n×m (10×10 como máximo) y posteriormente almacene toda la matriz en un vector. Imprimir el vector resultante.
5. Almacene los elementos de una matriz n×m (10×10 como máximo) y calcule el producto de dicha matriz por un vector.
6. Almacene los elementos de una matriz de 4×4 (números enteros positivos). Imprimir la fila que contiene el número menor y la columna que tenga el número mayor.
7. Leer los elementos de una matriz cuadrada m×n y almacenarlos los elementos en otra matriz donde se invertirá el orden original de cada fila. Al final imprimir la matriz resultante.
8. Leer los elementos de una matriz m×n donde no se deben ingresar números repetidos.
9. Leer los elementos de una matriz n×m (máximo 10×10) y generar un vector que almacene el máximo elemento de cada columna de la matriz. Imprimir dicho vector.
10. Genere una matriz cuadrada (máximo 10×10) con números aleatorios entre 1 y 50, manejando el siguiente criterio: las filas se llenarán con números acordes a ellas, si son filas impares (1,3,5,etc.) con números impares y si son las filas pares (0,2,4,etc.) con números pares. Al final imprimir dicha matriz.
11. Leer los elementos de una matriz cuadrada (máximo 15×15) y determinar si la matriz cumple con alguna de las siguientes propiedades:

    a) Matriz simétrica (si los valores de cada fila son iguales a los de su columna correspondiente)

    b) Matriz identidad ($a_{ij} = 0$ si $i \neq j$ y $a_{ij} = 1$ si $i = j$)

    c) Matriz triangular superior (todos los elementos por debajo de la diagonal son ceros)

    d) Matriz triangular inferior (todos los elementos por encima de la diagonal son ceros)

12. Leer los elementos de una matriz cuadrada (máximo 15×15) e imprimir :

    a) La traspuesta

    b) La suma de todos los elementos que no son de la "periferia" de la matriz.

13. Genere una matriz 4×4 con las primeras 16 letras del abecedario en minúsculas:

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

14. Leer los elementos de una matriz cuadrada de tamaño *n* (10×10 como máximo), calcular la sumatoria de los elementos de las diagonales (principal e inversa) y guardar estos elementos en dos vectores (DP y DI).





n = 4, sumatoria de DP=19 y sumatoria de DI=50.



M	0	1	2	3	*DP*	*DI*
0	7	1	9	6	7	6
1	4	2	20	7	2	20
2	3	22	1	8	1	22
3	2	11	10	9	9	2

15. Genere la matriz espiral (en forma de caracol) de n×n.

Ejemplos



n=5:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

16. Genere un cuadrado mágico 3 × 3 (matriz 3 × 3 formada por números del 1 al 9 donde la suma de sus filas, sus columnas y sus diagonales son idénticas).
17. Leer los elementos de una matriz de 3×3 y calcular su determinante.
18. Genere una matriz de 5×5 con números aleatorios entre 1 y 100; posteriormente ordenarla en forma descendente utilizando el método de la burbuja.
19. En el triángulo de Pascal cada número es la suma de los que se encuentran a ambos lados en el renglón de arriba. Calcular e imprimir los *n* primeros renglones del triángulo de Pascal. No es necesario que la impresión sea en la forma simétrica de triángulo:

    1
    1 1
    1 2 1
    1 3 3 1
    1 4 6 4 1

20. En el campeonato de futbol mexicano participan *n* equipos. Cada equipo se identifica con un número. Para cada equipo se registra el número de:

    a) Partidos ganados (PG).
    b) Partidos empatados (PE).
    c) Partidos perdidos (PP).
    d) Partidos jugados (PJ).
    e) Diferencia de goles (DG).

    En la última columna calcular los puntos respectivos sabiendo que se obtienen: 3 puntos por partido ganado y 1 punto por empate. Imprimir una matriz con los datos obtenidos por cada equipo. Al final imprimir el número del equipo campeón.
    Para el desempate en el primer lugar se toma en cuenta:

    a) Mayor número de partidos ganados.
    b) Mejor diferencia de goles.

21. Considere el siguiente listado de países y capitales: Alemania-Berlín, Argentina-Buenos Aires, Brasil-Brasilia, Canadá-Ottawa, Chile-Santiago, Ecuador-Quito, Inglaterra-Londres, Francia-París, Italia-Roma, Japón-Tokio, México-D.F. Elabore un programa que permita leer el nombre de un país e imprimir el nombre de la capital o viceversa.



Capítulo 5



Programación modular

Al término de este capítulo, el alumno será capaz de

- Definir y utilizar la técnica de dividir un problema en módulos para facilitar su resolución y mantenimiento así como la reutilización del código.

Contenido

5.1 Definición

Un problema difícil es más sencillo al dividirlo en pequeñas partes y tratar de buscar la solución de cada una de ellas y así resolver todo el problema general; lo mismo sucede en programación, la mejor forma de elaborar y dar mantenimiento a un programa complejo es construirlo a partir de bloques menores o módulos, pues de esta manera es más fácil analizar, programar y darle mantenimiento a cada uno de ellos, que si estuviera todo el código en el programa principal. Dichos módulos se escriben solamente una vez, pero pueden ser llamados en diferentes puntos del programa principal o de cualquier otro módulo.

Al método que plantea la solución de un problema principal partiendo de las soluciones de sus subproblemas se le conoce como *diseño descendente*.[1] Se denomina así ya que inicia en la parte superior con un problema general y el diseño específico de las soluciones de los subproblemas.

La programación modular es una técnica que consiste en dividir un programa en tareas y dar origen a la creación de pequeños programas llamados *módulos, subprogramas o subrutinas* con la finalidad de simplificar la elaboración y mantenimiento del mismo (véase figura 5.1), donde cada módulo se codifica y se procesa de manera independiente, sin importar los detalles de otros módulos. Esto facilita la localización de un error cuando se produce. Este tipo de programación es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa.

En lenguaje C a cada módulo o subprograma se le conoce como función; en este lenguaje se trabaja a base de funciones y, de manera general, los programas se elaboran combinando funciones que el programador escribe y funciones "predefinidas" disponibles en la biblioteca estándar de C.

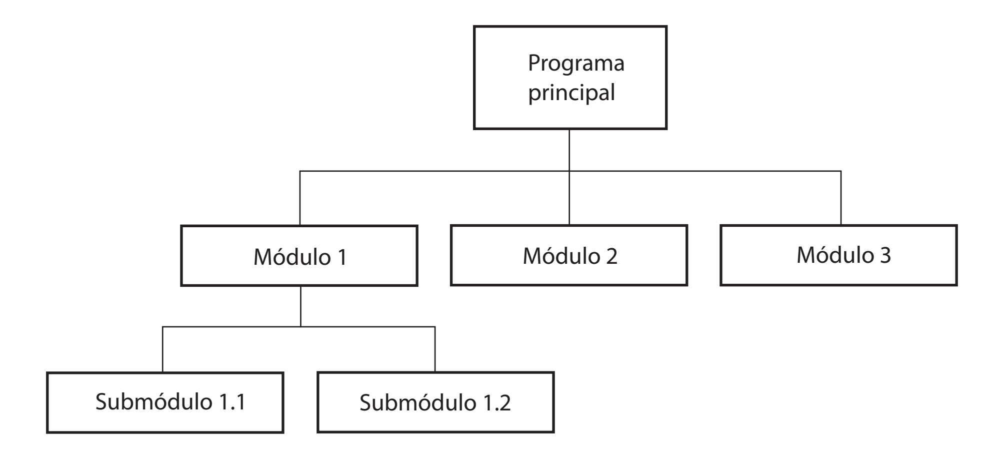


Figura 5.1 Esquema de programación modular.

Las ventajas de la programación modular son las siguientes:

1. Facilita el diseño descendente.
2. Se simplifica un algoritmo complejo.
3. Cada módulo se puede elaborar de manera independiente, lo que permite trabajar simultáneamente a varios programadores y con ello disminuir el tiempo de elaboración del algoritmo.
4. La depuración se lleva a cabo en cada módulo.
5. El mantenimiento es más sencillo.
6. Creación de bibliotecas con módulos específicos (se pueden utilizar en otros programas).

Nota: En este capítulo a los módulos o subprogramas los llamaremos *funciones*, ya que todas las características descritas son propias del lenguaje C.[2]

---

[1] Diseño descendente *(top-down)* es un proceso mediante el cual un problema se descompone en un serie de niveles o pasos sucesivos de refinamiento *(stepwise).*

[2] Cada lenguaje de programación tiene sus propias características en programación modular; en este caso de lenguaje C.

5.2 Programa principal y funciones

La base de la programación en C es la función, ya que constituye una parte fundamental de la codificación en el proceso de solución de problemas. Un programa contiene una o más funciones en uno o más archivos. Una de las funciones es *main()*, donde se inicia la ejecución del programa.

El resto de las funciones se llaman desde *main()* y desde el interior de otras funciones.

5.2.1 Programa principal o función *main()*

El papel más importante del *programa principal (main())* es coordinar a las otras funciones mediante llamadas o invocaciones.

5.2.2 Función

Es un subprograma que realiza una tarea específica que puede o no recibir valores (parámetros). En C podemos devolver cualquier tipo de datos escalares (puntero, tipo numérico y el tipo carácter o en su caso regresar un valor nulo que llamaremos *nada* o *ninguno*). Asimismo, no se pueden devolver arreglos ni estructuras. El uso de funciones es una práctica común y recomendable ya que permite dividir el código, simplificando así el desarrollo y la depuración del mismo. Para utilizar funciones en un programa es necesario declararlas previamente.

5.2.3 Ámbito de las variables

Variable local. Variable declarada en una determina función, sólo se encuentra disponible durante el funcionamiento de la misma, es decir está en memoria cuando dicha función está activa.

Variable global.[3] Variable declarada fuera de cualquier función y que puede ser utilizada por las funciones que se encuentran después de dicha declaración; por lo tanto si la declaramos junto a las librerías, la podrá utilizar todo el programa. Esta características es propia del lenguaje C.

5.2.4 Llamada o invocación de una función

Para llamar una función es necesario escribir el nombre o identificador de la función y en seguida un par de paréntesis, que estarán vacíos si hablamos de funciones sin paso de parámetros y tendrán información en caso de ser funciones con paso de parámetros.

5.2.5 Cómo se ejecuta un programa que contiene funciones

El compilador siempre inicia con la primera instrucción que encuentra en el programa principal (main) y continúa ejecutándolo según la estructura de control (secuencial, selectiva o repetitiva) del programa; cuando encuentra la llamada a una función éste se traslada y ejecuta el cuerpo de la función desde la primera instrucción hasta encontrar el fin de la misma; cuando esto sucede el compilador regresa a la siguiente instrucción que llamó a la función y continúa de la misma manera.

5.2.6 Ubicación de una función en un programa

Las funciones pueden ir antes o después del programa principal, pero si se ubican después es necesario escribir el *prototipo* de la función (el encabezado) antes del programa principal. Una función puede que regrese o no regrese nada (*void*). En el primer caso se comporta como una variable, ya que la podemos imprimir, asignar a otra variable, comparar, utilizar como parámetro, etc. En el segundo caso, el resultado lo tenemos que imprimir dentro de la función (sólo las variables globales se imprimirán fuera). Existen dos tipos de

[3] Es importante que la comunicación entre un programa y un subprograma o entre subprogramas deba realizarse a través de parámetros, y no de variables globales, ya que consumen más memoria.

funciones: Sin paso de parámetros y con paso de parámetros. A partir de la sección 5.4 mencionaremos cada una de ellas.

5.3 Prototipo de una función

Un prototipo es el encabezado de una función, es decir la primera línea de la función. La ventaja de utilizar prototipos es que las funciones pueden estar en cualquier lugar y en cualquier orden y pueden ser llamadas desde cualquier punto del programa principal o de otra función.

En lenguaje C, lleva punto y coma al finalizar. Primero mostramos la sintaxis de funciones sin paso de parámetros y sin prototipos:

Ejemplos



Pseudocódigo	Lenguaje C
	`# librerías`
Variables globales	`Variables globales;`
Constantes	`Constantes`
nada funcion_a ( )	`void funcion_a ( )`
inicio	`{`
variables locales	`variables locales;`
Instrucción 1	`Instrucción 1;`
Instrucción 2	`Instrucción 2;`
fin	`}`
nada funcion_b ( )	`void funcion_b ( )`
inicio	`{`
variables locales	`variables locales;`
Instrucción 1	`Instrucción 1;`
Instrucción 2	`Instrucción 2;`
fin	`}`
principal ( )	`main ( )`
inicio	`{`
Variables locales	`Variables locales;`
Cuerpo del programa principal	`Cuerpo del programa principal`
::::::::::::::::::	`::::::::::::::::`
funcion_a( )	`funcion_a( );`
funcion_b( )	`funcion_b( );`
Instrucción 3	`Instrucción 3;`
:::::::::::::::::::	`:::::::::::::::::::`
fin	`}`

Si se desea declarar primero los prototipos de cada función, luego el cuerpo de la función principal y al final el cuerpo de cada función, ésta sería su sintaxis u orden:

Pseudocódigo	Lenguaje C
	`# librerías`
Variables globales	`Variables globales`
Constantes	`Constantes`
funcion_a ()// prototipo de la función a	`funcion_a ();// prototipo de la función a`
funcion_b ()// prototipo de la función b	`funcion_b ();// prototipo de la función b`
principal ()	`main()`
inicio	`{`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
Variables locales	`Variables locales`
Cuerpo del programa principal	`Cuerpo del programa principal`
::::::::::::::::::	`:::::::::::::::::::`
funcion_a()	`funcion_a();`
funcion _b()	`funcion_b();`
::::::::::::::::::	`:::::::::::::::::::`
fin	`}`
funcion _a()	`funcion_a ()`
inicio	`{`
Variables locales	`Variables locales`
Instrucción 1	`Instrucción 1;`
Instrucción 2	`Instrucción 2;`
Instrucción 3	`Instrucción 3;`
fin	`}`

Pseudocódigo	Lenguaje C
funcion_b()	`funcion_b ();`
inicio	`{`
Variables locales	`Variables locales`
Instrucción 1	`Instrucción 1;`
Instrucción2	`Instrucción 2;`
fin	`}`

La desventaja de las funciones sin prototipos, es que en el momento de llamarlas o invocarlas tienen que estar declaradas antes; por ejemplo, en la sintaxis de las funciones sin paso de parámetros y sin prototipos la *funcion_b ( )* y *principal ( ) (main)* llaman a la *funcion_a ( )* que está declarada antes; sin embargo la *funcion_a ( )* no puede llamar a la *funcion_b ( )*. La *función main()* puede llamar a cualquier función, pero ninguna otra función puede llamar a main().

Un *prototipo* sin paso de parámetros es el encabezado de la función; en lenguaje C, como veremos en seguida, a diferencia de la declaración de una función, lleva punto y coma. La ventaja es que le avisa al compilador que existe una función que regresa un determinado tipo de dato y puede ser utilizada en cualquier momento, por lo que la declaración de las funciones no tiene que ser antes de la llamada sino después de la función principal o main.

5.4 Funciones sin paso de parámetros



Son aquellas que no reciben parámetros o valores, ya que éstos se solicitan dentro de la función, luego se realizan las instrucciones (cálculos u operaciones) y normalmente se imprime el resultado.

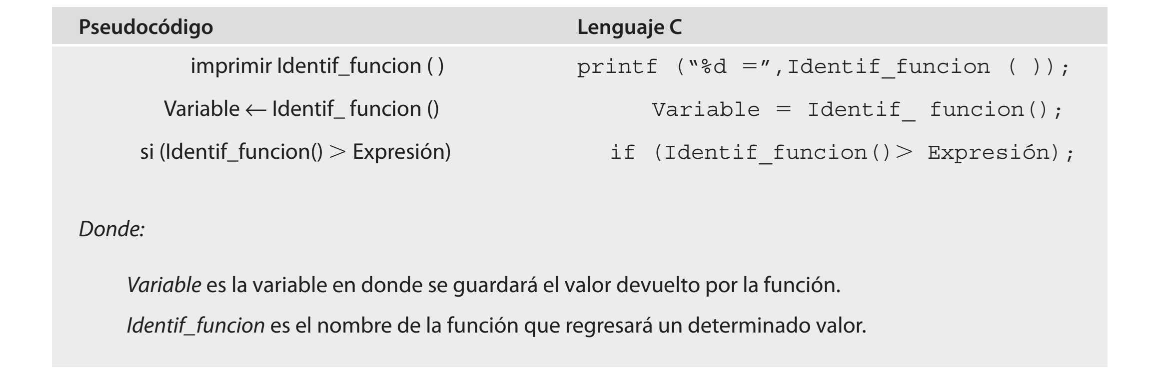
A continuación aparece la *sintaxis* correspondiente:

Pseudocódigo	Lenguaje C
nada Identif_funcion( )	`void Identif_funcion ( )`
inicio	`{`
Declaración de variablaes	`Declaración de variables;`
Cuerpo de la función	`Cuerpo de la función;`
::::::::::::::::::::	`::::::::::::::::::::`
::::::::::::::::::::	`::::::::::::::::::::`
fin	`}`

De llamadas a una función sin paso de parámetros:

Ejemplos





Pseudocódigo	Lenguaje C
imprimir Identif_funcion ( )	`printf (“%d =”,Identif_funcion ( ));`
Variable ← Identif_ funcion ()	`Variable = Identif_ funcion();`
si (Identif_funcion() > Expresión)	`if (Identif_funcion()> Expresión);`

Donde:

Variable es la variable en donde se guardará el valor devuelto por la función.
Identif_funcion es el nombre de la función que regresará un determinado valor.

5.5 Ejercicios resueltos de funciones sin paso de parámetros

Ejercicio 1. Imprimir una serie de enunciados según el orden del programa principal y las funciones.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
nada funcion_x ( )	`void funcion_x ( )`
inicio	`{`
imprimir “Programación modular”	`printf( “Programación modular\n”);`
imprimir “ Las funciones son importantes”	`printf( “Las funciones son importantes\n”);`
fin	`}`
nada principal ( )	`void* main( )`
inicio	`{`
imprimir “Modulo 5”	`printf( “Modulo 5\n”);`
funcion_x ( )	`funcion_x ( );`
imprimir “y nos ayudan en la programación”	`printf( “y nos ayudan en la programación\n”);`
	`getch(); return 0;`
fin	`}`

* Si se utiliza el compilador DEV-CPP y Code::Blocks, no se puede utilizar el tipo de dato *void*; en su lugar se utiliza el tipo *int* o se omite el tipo.

Todo programa inicia con la primera instrucción del programa principal; en este ejercicio es *imprimir “Módulo 5”,* así que el compilador la ejecuta y muestra en pantalla *Módulo 5*; la siguiente instrucción es la llamada a la función *funcion_x()*, así que el compilador se ubica en la primera instrucción de la *funcion_x*, que en este caso es: *imprimir “Programación modular”*, así que el compilador la ejecuta y se muestra en pantalla *Programación modular*; se continúa con la siguiente instrucción de la *funcion_x* que es *imprimir “Las funciones son importantes”,* así que el compilador la ejecuta y se muestra en pantalla *Las funciones son importantes*, luego se encuentra el fin de la *funcion_x*, por consiguiente el compilador regresa a la siguiente instrucción del programa que lo llamó, que en este caso es *imprimir “y nos ayudan en la programación”,* por lo tanto la última línea que se imprime es: *y nos ayudan en la programación*. Por último encontramos el fin del programa principal y con eso termina el mismo.

Ejercicio 2. Imprimir una serie de enunciados según el orden en que las funciones son llamadas.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
nada funcion_a ( )	`void funcion_a ( )`
inicio	`{`
imprimir "Introducción a la programación"	`printf ("Introducción a la programación\n");`
fin	`}`
nada funcion_b ( )	`void funcion_b ( )`
inicio	`{`
imprimir "Introducción a la física"	`printf ("Introducción a la física\n");`
fin	`}`
nada funcion_c ( )	`void funcion_c ( )`
inicio	`{`
imprimir "Introducción a la química"	`printf ("Introducción a la química\n") ;`
fin	`}`
nada funcion_d ( )	`void funcion_d ( )`
inicio	`{`
imprimir "Introducción a la computación"	`printf ("Introducción a la computación\n");`
fin	`}`
principal ()	`main()`
inicio	`{`
funcion_a( )	`funcion_a( );`
funcion_b( )	`funcion_b( );`
funcion_c( )	`funcion_c( );`
funcion_d()	`funcion_d( );`
	`getch(); return 0;`
fin	`}`

Este programa consta de cinco funciones, una de las cuales es la función principal que llama a las otras cuatro. La primera instrucción del programa principal es la llamada a la funcion_a, en la cual su primera y única instrucción es *imprimir "Introducción a la programación"*; el compilador la ejecuta y muestra en pantalla *Introducción a la programación*, encuentra el fin de la funcion_a y el compilador regresa al programa principal donde la siguiente instrucción es la llamada a la funcion_b, la cual ejecuta e imprime *Introducción a la física*; regresa al programa principal donde la siguiente instrucción es la llamada a la *funcion_c*, la cual ejecuta e imprime *Introducción a la química*; regresa al programa principal donde la siguiente instrucción es la llamada a la *funcion_d*, la cual ejecuta e imprime *Introducción a la computación*, regresa al programa principal y se encuentra con el fin del programa.

Ejercicio 3. Utilización de las mismas funciones del ejercicio 2 pero con el siguiente programa principal.

Pseudocódigo	Lenguaje C
principal ( )	`main( )`
inicio	`{`
imprimir " Materias de la carrera"	`printf (" Materias de la carrera\n");`
funcion_b( )	`funcion_b( );`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
funcion_d( )	`funcion_d( );`
funcion_a( )	`funcion_a( );`
funcion_c( )	`funcion_c( );`
imprimir"Otras más"	`printf ("Otras más\n");`
funcion_d( )	`funcion_d( );`
funcion_a( )	`funcion_a( );`
	`getch(); return 0;`
fin	`}`

Es similar al ejercicio 2, sólo que la primera instrucción del programa es imprimir "Materias de la carrera", luego llama a las cuatro funciones en el siguiente orden: *funcion_b, funcion_d, funcion_a, funcion_c*, e imprime *Introducción a la física, Introducción a la computación, Introducción a la programación, Introducción a la química*; luego regresa al programa principal y la siguiente instrucción es imprimir "Otras más" y llama otra vez a la *funcion_d* mostrando en pantalla *Introducción a la computación*, luego regresa al programa principal y por último llama a la funcion_a e imprime *Introducción a la programación.*

Como se puede apreciar, es posible llamar más de una función según la lógica del programa; en las funciones con paso de parámetro se aprecia mejor la utilización de las funciones.

Ejercicio 4. Imprimir una serie de enunciados según el orden en que las funciones son llamadas.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
nada funcion_2 ( )	`void funcion_2 ( );`
nada funcion_1 ( )	`void funcion_1 ( )`
inicio	`{`
imprimir "Azul"	`printf ("Azul\t");`
imprimir "Verde"	`printf ("Verde\t");`
fin	`}`
nada funcion_3 ( )	`void funcion_3 ( )`
inicio	`{`
imprimir "Blanco"	`printf ("Blanco\t");`
imprimir "Negro"	`printf ("Negro\t");`
fin	`}`
principal ( )	`main( )`
inicio	`{`
imprimir "Los colores"	`printf ("Los colores\n");`
imprimir "Lila"	`printf ("Lila\t");`
funcion_2 ( )	`funcion_2 ( );`
imprimir "café"	`printf ("café\t");`
	`getch(); return 0;`
fin	`}`
nada funcion_2 ( )	`void funcion_2 ( )`
inicio	`{`
imprimir "Amarillo"	`printf ("Amarillo\t");`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
imprimir "Rojo"	`printf ("Rojo\t");`
funcion_1 ( )	`funcion_1 ( );`
imprimir "Naranja"	`printf ("Naranja\t");`
fin	`}`

En este ejercicio comenzamos a utilizar el concepto de prototipo, en la primera línea del pseudocódigo aparece el prototipo nada funcion_2, cuyo equivalente en lenguaje C es void funcion_2;. Es importante recordar que al final de un prototipo en C lleva punto y coma. En este ejemplo la funcion _2 se declaró después de principal o en el caso del lenguaje C de main, ya que el prototipo va y busca en qué parte del programa está declarada dicha función. Si no hubiéramos declarado el prototipo en el momento de llamar a la función funcion_2, marcaría error, ya que el compilador la buscaría arriba y no la encontraría. En este ejercicio vemos la desventaja de *no* utilizar prototipos ya que funcion_1 no puede llamar a ninguna otra función.

Este ejercicio nos muestra que una función puede ser llamada por el programa principal o por otra función; si compilamos el programa la primera instrucción es *imprimir "Los colores"*, se muestra en pantalla *Los colores* y *Lila*, luego llama a la funcion_2 cuya primera instrucción es *imprimir "Amarillo"*, muestra en pantalla *Amarillo*; luego la siguiente instrucción es *imprimir "Rojo"* y muestra en pantalla *Rojo*, la siguiente instrucción llama a la funcion_1 cuya primera línea *imprime "Azul"*, muestra en pantalla *Azul*, después *imprime "Verde"*, encuentra el final de la funcion_1 y regresa a la siguiente que *imprime "Naranja"*, encuentra el fin de la funcion_2 y regresa al programa principal para imprimir café. Como vemos, ninguna función llama a funcion_3.

En este ejercicio vemos la desventaja de *no* utilizar prototipos ya que funcion_1 ( ) no puede llamar a ninguna otra función.

Ejercicio 5. Haga el programa principal y dos funciones sin paso de parámetros: *a*) sumar dos números enteros y *b*) multiplicar dos números enteros.

Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
nada suma ( )	`void suma ( );`
nada producto ( )	`void producto( )`
inicio	`{`
entero n1, n2, prod	`int n1, n2, prod;`
imprimir "Dame el primer número entero "	`printf ("Dame el primer número entero ");`
leer n1	`scanf ("%d",&n1);`
imprimir "Dame el segundo número entero "	`printf ("Dame el segundo número entero ");`
leer n2	`scanf ("%d",&n2);`
prod ← n1 * n2	`prod = n1 * n2  ;`
imprimir "La multiplicación es: ", prod	`printf ("La multiplicación es: %d",prod);`
fin	`}`
principal ( )	`main()`
inicio	`{`
entero opc	`int opc;`
imprimir "¿Qué deseas hacer?"	`printf ("¿Qué deseas hacer?\n");`
imprimir "1. Suma de dos números "	`printf ("1. Suma de dos números\n");`
imprimir "2. Producto de dos números "	`printf ("2. Producto de dos números\n");`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
leer opc	`scanf ("%d",&opc);`
si (opc = 1) suma ( )	`if (opc == 1) suma ( );`
si no producto( )	`else producto( );`
	`getch();`
	`return 0;`
fin	`}`
nada suma ( )	`void suma( )`
inicio	`{`
entero n1, n2, suma	`int n1, n2,suma;`
imprimir "Dame el primer número entero "	`printf ("Dame el primer número entero ");`
leer n1	`scanf ("%d",&n1);`
imprimir "Dame el segundo número entero "	`printf ("Dame el segundo número entero ");`
leer n2	`scanf ("%d",&n2);`
suma ← n1 + n2	`suma =n1 + n2;`
imprimir "La suma es: ", suma	`printf ("La suma es: %d",suma);`
fin	`}`

En la primera línea del pseudocódigo aparece el prototipo nada suma, cuyo equivalente en lenguaje C es void suma. Al final de un prototipo en C lleva punto y coma. En este ejemplo la suma se declaró después de principal o en el caso del lenguaje C de main, ya que el prototipo va y busca en qué parte del programa está declarada dicha función. Si no hubiéramos declarado el prototipo en el momento de llamar a la función suma, marcaría error, ya que el compilador la buscaría arriba y no la encontraría. El programa principal tiene una variable opcional *opc* de tipo entero, y por lo tanto ésta queda en memoria para su utilización, luego se *imprime "¿Qué deseas hacer?"* y se muestran dos opciones: *a*) Suma de dos números, y *b*) Producto de dos números; el usuario elige una opción (1 o 2) y se almacena en la variable *opc*. Si eligió opc=1 llama a la función suma, la cual declara tres variables locales de tipo entero *n1*, *n2* y *suma*, solicita dos números y los almacena en *n1* y *n2*, luego hace la suma y almacena el resultado en la variable *suma* e imprime el resultado *"La suma es: "*. Si eligió opc=2 llama a la función producto, la cual declara tres variables locales de tipo entero n1, n2 y prod, solicita dos números y los almacena en *n1* y *n2*, luego hace la multiplicación y almacena el resultado en la variable *prod* e imprime el resultado, *"El producto es: ". Por lo tanto el programa principal solamente llama a una función cada vez que se corre el mismo.* Hay que recordar que en las funciones que no regresan nada (void) el resultado se imprime dentro de la función.

Ejercicio 6. Haga el programa principal y opciones a tres funciones sin paso de parámetros que calculen: a) La hipotenusa de un triángulo rectángulo, b) El área de un triángulo, c) El tercer ángulo de un triángulo.

Pseudocódigo
nada hipotenusa ( )
inicio
entero co, ca
real h
imprimir "Escriba la longitud del cateto adyacente al ángulo:"
leer ca
imprimir "Escriba la longitud del cateto opuesto al ángulo: "
leer co h ← raizcuad (co * co + ca * ca)

(*continúa*)

(*continuación*)

Pseudocódigo

```
    imprimir "La hipotenusa del triángulo mide: ", h
fin
nada area ( )
inicio
    entero base, alt
    real a
    imprimir "Escriba la base y la altura:"
    leer base, alt
    a ← base * alt / 2
    imprimir "El área del triángulo es: ", a
fin
nada angulo ( )
inicio
    real a1, a2, a3
    imprimir "Escriba el valor de los dos ángulos:"
    leer a1, a2
    a3 ← 180 − ( a1 + a2)
    imprimir "El tercer ángulo mide: ", a3
fin
principal ( )
inicio
    caracter opc
    imprimir  "a) Hipotenusa de un triángulo rectángulo"
    imprimir  "b) Área de un triángulo"
    imprimir  "c) Tercer ángulo de un triángulo"
    imprimir  "Elige una opción"; leer opc
    segun_sea (opc)
    inicio
        caso 'a' : hipotenusa( )
                salir
        caso 'b' : area( )
                salir
        caso 'c' : angulo( )
                salir
    fin
fin
```

El programa no tiene prototipos, por ello las funciones se declararon antes de ser llamadas, en este caso antes del programa principal (main); tiene una función principal y 3 funciones, cada una de ellas con sus variables locales y su cuerpo de la función; el programa principal declara una variable local *opc* de tipo carácter para almacenar a, b o c según lo que el usuario elija; se utilizan las estructuras de control selectiva múltiple ya que son tres opciones diferentes. Si elige la opc=a llamará a la función hipotenusa, si elige la opc=b llamará a la función area y si elige la opc=c llamará a la función angulo. Cada función es independiente y puede ser llamada en cualquier orden. Cuando se ejecuten alguna de las tres funciones y encuentre el final de la misma regresa al programa principal, que en este caso en el fin del segun_sea, y concluye el programa.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void hipotenusa ( )
{
   int co, ca;
   float h;
   printf (”Escriba la longitud del cateto adyacente al ángulo: ”);
   scanf (“%d”,&ca);
   printf (“Escriba la longitud del cateto opuesto al ángulo: “);
   scanf (“%d”,&co);
   h = sqrt (co * co + ca * ca);
   printf (“La hipotenusa del triángulo mide: %f“, h);
}
void area ( )
{
   int base, alt; float a;
   printf (”Escriba la base y la altura: ”);
   scanf (“%d%d”,&base, &alt);
   a = base * alt / 2;
   printf (“El área del triángulo es: %f“, a);
}
void angulo ( )
{
   float a1, a2, a3;
   printf (”Escriba el valor de los dos ángulos: ”);
   scanf (“%f%f”,&a1, &a2);
   a3 = 180 – (a1+a2) ;
   printf (“El tercer ángulo mide: %f“, a3);
}
main( )
{  char opc;
   printf (“a)Hipotenusa de un triángulo rectángulo \n“);
   printf (“b)Área de un triángulo\n“);
   printf (“c)Tercer ángulo de un triángulo \n“);
   printf (“Elige una opción\n“); scanf (“%c”,&opc);
   switch (opc)
   {
        case ‘a’:hipotenusa( );
                 break;
        case ‘b’:area( );
                 break;
        case ‘c’:angulo( );
                 break;
   }
   getch(); return 0;
}
```

Ejercicio 7. Haga el programa principal con opciones a cuatro funciones sin paso de parámetros que conviertan de: a) Grados centígrados a grados Fahrenheit, b) Grados Fahrenheit a grados centígrados, c) Grados centígrados a kelvins, d) Kelvins a grados centígrados.

Pseudocódigo

```
nada convertir_C_F ( )
nada convertir_F_C ( )
nada convertir_C_K ( )
nada convertir_K_C ( )
principal ( )
inicio
    caracter opc
    hacer
    inicio
        imprimir   "Elige una de las siguientes conversiones"
        imprimir   "a) Grados centígrados a grados Fahrenheit "
        imprimir   "b) Grados Fahrenheit a grados centígrados "
        Imprimir   "c) Grados centígrados a kelvins"
        Imprimir   "d) Kelvins a grados centígrados"
        leer opc
        segun_sea (opc)
        inicio
            caso 'a' :  convertir_C_F( );  salir
            caso 'b' :  convertir_F_C( );  salir
            caso 'c' :  convertir_C_K( );  salir
            caso 'd' :  convertir_K_C( );  salir
        fin
        imprimir"¿Deseas repetir el programa S/N? ";   leer opc
    fin
    mientras (opc = 'S')
fin
nada convertir_C_F ( )
inicio
real c , f
    imprimir"Dame los grados centígrados:" ; leer c
    f ← 1.8*c +32
    imprimir "Los grados son: ", f
fin
nada convertir_F_C ( )
inicio
    real c , f
    imprimir"Dame los grados Fahrenheit:" ; leer f
    c ← ( f −32 ) * 5/9
    imprimir "Los grados son: ", c
fin
nada convertir_C_K ( )
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
inicio
    real c , k
    imprimir"Dame los grados centígrados:"; leer c
    k ← c + 273
    imprimir "Los grados son: ", k
fin
nada convertir_K_C ( )
inicio
    real c , k
    imprimir"Dame los kelvins:"; leer k
     c ← 273 − k; imprimir "Los grados son: ", c
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
void convertir_C_F ( );
void convertir_F_C ( );
void convertir_C_K ( );
void convertir_K_C ( );
main( )
{
    char opc;
    do
    {
        printf ("Elige una de las siguientes conversiones \n");
        printf ("a)Grados centígrados a grados Fahrenheit \n");
        printf ("b)Grados Fahrenheit a grados centígrados \n");
        printf ("c)Grados centígrados a Kelvins\n");
        printf ("d)Kelvins a grados centígrados\n");
        scanf ( "%s",&opc);
        switch (opc)
        {
               case 'a' : convertir_C_F( ); break;
               case 'b' : convertir_F_C( ); break;
               case 'c' : convertir_C_K( ); break;
               case 'd' : convertir_K_C( ); break;
        }
        printf ("\n¿ Deseas repetir el programa S/N?\n"); scanf ( "%s",&opc);
    }while (opc == 'S'); return 0;
}
void convertir_C_F ( )
```

(*continúa*)

(*continuación*)

Lenguaje C

```
{
    float c , f ;
    printf (”Dame los grados centígrados: ” ); scanf (“%f”,&c);
    f = 1.8 *c + 32; printf (“Los grados son: %f“, f);
}
void convertir_F_C ( )
{
    float c , f ;
    printf (”Dame los grados Fahrenheit: ” ); scanf (“%f”,&f); c =(f−32 ) * 5/9;
    printf (“Los grados son: %f“, c);
}
void convertir_C_K ( )
{
    float c , k ;
    printf (”Dame los grados centígrados: ” ); scanf (“%f”,&c);
    k = c + 273; printf (“Los grados son: %f“, k);
}
void convertir_K_C ( )
{
    float c , k ;
    printf (”Dame los kelvins: ” ); scanf (“%f”,&k);
    c = 273 − k; printf (“Los grados son: %f“, c);
}
```

Este ejercicio tiene prototipos, por ello las funciones se declararon después de ser llamadas (después de la función main); tiene una función principal y cuatro funciones para convertir de: a) Grados centígrados a grados Fahrenheit, b) Grados Fahrenheit a grados centígrados, c) Grados centígrados a kelvins, y d) Kelvins a grados centígrados. El programa principal tiene una estructura de control selectiva múltiple para cada opción. Cada función tiene declaradas sus variables locales, sus operaciones en el mismo cuerpo de la función y la impresión del resultado. El programa principal incorpora una *hacer_mientras*, para que el usuario tenga la opción de repetir el programa y elegir la misma u otra opción las veces que lo requiera, siempre y cuando conteste *S* para repetir el programa.

Ejercicio 8. Haga el programa principal con opciones a cinco funciones sin paso de parámetros que calcule las siguientes áreas: a) Círculo, b) Cuadrado, c) Rectángulo, d) Triángulo, e) Trapecio.

Pseudocódigo

```
constante PI ← 3.1416
nada area_circulo ( )
inicio
    entero radio; real area_cir
    imprimir”Escriba el radio:”; leer radio
    area_cir ← PI * radio * radio
    imprimir “El área del círculo es: “, area_cir
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
fin
nada area_cuadrado ( )
inicio
    entero lado, area_cua
    imprimir"Escriba un lado:" ; leer lado
    area_cua ← lado * lado
    imprimir "El área del cuadrado ", area_cua
fin
nada area_rectangulo ( )
inicio
    entero base, alt, area_rec
    imprimir "Escriba la base y la altura:" ; leer base, alt
    area_rec ← base * alt
    imprimir "El área del rectángulo es: ", area_rec
fin
nada area_triangulo ( )
inicio
    entero base, alt; real area_tria
    imprimir"Escriba la base y la altura:" ; leer base, alt
    area_tria ← base * alt / 2
    imprimir "El área del triángulo es: ", area_tria
fin
nada area_trapecio ( )
inicio
    entero basemay, basemen, alt;   real area_tra
    imprimir"Escriba la base mayor la base menor y la altura:" ; leer basemay, basemen , alt
    area_tra ← (basemay + basemen) * alt / 2
    imprimir "El área del trapecio es: ", area_tra
fin
principal ( )
inicio
    entero opc
    hacer
    inicio
        imprimir   "Elige una de las siguientes áreas"
        imprimir   "1)Círculo "
        imprimir   "2)Cuadrado "
        Imprimir   "3)Rectángulo"
        Imprimir   "4)Triángulo"
        Imprimir   "5)Trapecio"; leer opc
        segun_sea (opc)
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
            inicio
                  caso 1 : area_circulo( ); salir
                  caso 2 : area_cuadrado( ); salir
                  caso 3 : area_rectangulo( ); salir
                  caso 4 : area_triangulo( ); salir
                  caso 5 : area_trapecio( ); salir
            fin
            imprimir"¿Deseas repetir el programa S=1 / N=2? "; leer opc
      fin
      mientras ( opc = 1)
fin
```

Lenguaje C

```c
#include <stdio.h>
#include <conio.h>
#define PI 3.1416
void area_circulo ( )
{
    int radio;
    float area_cir;
    printf (”Escriba el radio: ”); scanf (“%d”,&radio);
    area_cir = PI * radio * radio;
    printf (“El área del círculo es: %f“, area_cir);
}
void area_cuadrado ( )
{
    int lado, area_cua;
    printf (”Escriba un lado :”); scanf (“%d”,&lado);
    area_cua = lado * lado;
    printf (“El área del cuadrado %d“, area_cua);
}
void area_rectangulo ( )
{
    int base, alt, area_rec;
    printf (”Escriba la base y la altura : ”); scanf (“%d%d”,&base,&alt);
    area_rec = base * alt;
    printf (“El área del rectángulo es: %d“, area_rec);
}
void area_triangulo ( )
{
    int base, alt;
```

(*continúa*)

(*continuación*)

Lenguaje C

```
    float area_tria;
    printf (”Escriba la base y la altura: ”); scanf (“%d%d”,&base,&alt);
    area_tria = base * alt / 2;
    printf (“El área del triángulo es: %f“, area_tria);
}
void area_trapecio ( )
{
    float basemay, basemen, alt;
    float area_tra;
    printf (”Escriba la base mayor la base menor y la altura: ”);
    scanf (“%f%f%f”,&basemay,&basemen,&alt);
    area_tra = (basemay + basemen) * alt / 2;
    printf (“El área del trapecio es: %f“, area_tra);
}
main( )
{   int opc;
    do
    {
          printf (“Elige una de las siguientes áreas\n”);
          printf (“1)Círculo \n”);
          printf (“2)Cuadrado \n”);
          printf (“3)Rectángulo\n”);
          printf (“4)Triángulo\n”);
          printf (“5)Trapecio\n”);
          scanf (“%d”,&opc);
          switch (opc)
          {  case 1 : area_circulo( ); break;
             case 2 : area_cuadrado( ); break;
             case 3 : area_rectangulo( ); break;
             case 4 : area_triangulo( ); break;
             case 5 : area_trapecio( ); break;
          }
          printf (“\n¿Deseas repetir el programa S=1 / N=2? \n”);
          scanf (“%d”,&opc);
    } while ( opc == 1); return 0;
}
```

Este ejercicio no tiene prototipos. Tiene cálculos para cinco áreas definidas como: a) Círculo, b) Cuadrado, c) Rectángulo, d) Triángulo, e) Trapecio. Según la opción que desee el usuario, la variable *opc*, que es de tipo entero, almacena un número del 1 al 5, se realizará una de las funciones; al igual que en el ejercicio 7, el usuario tiene la posibilidad de repetir el programa las veces que desee, siempre y cuando elija S, es decir que capture el 1; cuando desee salirse del programa deberá elegir el 2, que es N.

5.6 Funciones con paso de parámetros

Estas funciones son las más utilizadas en la programación ya que pueden recibir uno o más valores llamados parámetros y regresan un solo valor de tipo entero, real o carácter. Si deseas regresar un arreglo de carácter es necesario hacerlo desde los parámetros. Los parámetros o valores son enviados del programa principal o de otra función. Dentro de la función se realizan solamente las instrucciones (cálculos u operaciones). Es importante revisar que el tipo de dato que regresará la función sea del mismo tipo que el valor declarado en el encabezado de la misma.

5.6.1 Parámetros de una función

También son llamados argumentos y se corresponden con una serie de valores que se especifican en la llamada a la función, o en la declaración de la misma, de los que depende el resultado de la función; dichos valores nos permiten la comunicación entre dos funciones. Los parámetros o argumentos de una función se pueden clasificar en:

1. *Formales o ficticios.* Son los que se encuentran en la definición de la función. Si una función usa argumentos, éstos se comportan como variables locales de la función. Son variables de la función que toman los valores que se le pasan desde fuera de ella. Veamos el siguiente bloque de programa:

```
float divide (int x, int y)
{
    int a, b;
    printf ("Dame dos números enteros: ");
    scanf ("%d%d",&a,&b);
```

   Tenemos una función llamada divide, la cual regresa un dato float y los parámetros formales o ficticios son la *x* y la *y*.

2. *Actuales o reales.* Se emplean en la llamada a la función. Pueden ser variables o expresiones cuyos valores se pasan a los argumentos formales. Algunos autores llaman argumentos a los parámetros actuales o reales. Veamos el siguiente bloque de programa:

```
main ()
{
    int m, n;
    float z;
    printf ("Dame dos números enteros: ");
    scanf ("%d%d",&m,&n);
    z = divide(m,n); //Llamada o invocación a la función divide
```

   En el renglón anterior los parámetros actuales o reales son la *m* y la *n*. Los parámetros formales siempre son variables, mientras que los reales pueden ser variables, constantes, expresiones aritméticas e incluso lo que regrese la llamada a otra función.

5.6.2 Paso de parámetros en una función

En C todos los parámetros se pasan *"por valor"*, es decir, en cada llamada a la función se genera una *copia* de los valores de los parámetros *actuales*, que se almacenan en variables temporales en la pila mientras dure la ejecución de la función. Sin embargo, cuando sea preciso es posible hacer que una función modifique el valor de una variable que se pase como parámetro *actual* en la llamada a la función. Para ello, lo que se debe proporcionar a la función no es el valor de la variable, sino su *dirección*, lo cual se realiza mediante un puntero que señale a esa dirección; a estos parámetros los llamamos *por variable* o *por referencia*; en este tipo de parámetros los datos salen modificados.

Nota: En este capítulo nos enfocaremos a los parámetros por valor, pero debemos aclarar que en el caso del paso de arreglos, éstos se comportan como parámetros por *variable* ya que existe un puntero que apunta a la primera posición.

A continuación veremos la sintaxis de las funciones con parámetros y sin prototipo.

Pseudocódigo	Lenguaje C
	# Librerías
Variables globales	Variables globales;
Constantes	Constantes
tipo_dato funcion_a (parámetros formales)[4]	tipo_dato funcion_a (parámetros formales)[4]
inicio	{
Variables locales	Variables locales;
Instrucción 1	Instrucción 1;
Instrucción 2	Instrucción 2;
Instrucción 3	Instrucción 3;
regresa (valor)[5]	return (valor)[5];
fin	}
tipo_dato funcion_b (parámetros formales)[4]	tipo_dato funcion_b (parámetros formales)[4]
inicio	{
Variables locales	Variables locales;
Instrucción 1	Instrucción 1;
c=funcion_a(parámetros actuales)[6]	C=funcion_a(parámetros actuales)[6];
Instrucción 2	Instrucción 2;
regresa (valor)[5]	return (valor)[5];
fin	}
principal ( )	main ( )
inicio	{
Variables locales	Variables locales;
Cuerpo del programa principal	Cuerpo del programa principal
::::::::::::::::::	::::::::::::::::::
c=funcion_a(parámetros actuales)[6]	C=funcion_a(parámetros actuales)[6];
d=funcion_b(parámetros actuales)[6]	D=funcion_b(parámetros actuales)[6];
Instrucción 3	Instrucción 3;
::::::::::::::::::::	::::::::::::::::::::
fin	}

En la sintaxis anterior las variables *c* y *d* son locales a la función respectiva; por lo tanto la *c* es diferente en la *funcion_b* que en el *main()*, ya que el ámbito de trascendencia lo tienen en la función correspondiente.

Como se comentó en la sección anterior, la desventaja de las funciones sin prototipos es que en el momento de llamarlas o invocarlas tienen que estar declaradas antes; por ejemplo, en la sintaxis anterior la *funcion_b* y *principal (main)* llaman a la *funcion_a*, que está declarada antes; sin embargo, la *funcion_a* no puede llamar a ninguna otra función.

[4] Cada parámetro debe tener su tipo y nombre; si son varios se separan con una coma.
[5] Es opcional que el valor vaya entre paréntesis.
[6] Nombre de cada parámetro; si son varios se separan con una coma.

La función *main()* puede llamar a cualquier función, pero ninguna otra función puede llamar a main().
Características de las funciones con paso de parámetros:

1. Normalmente el número de parámetros *actuales* es igual a los parámetros *formales*. Es decir que si en la llamada a la función se tienen dos parámetros, en la declaración de la función misma deberá tener dos parámetros. La excepción es cuando utilizamos los tres puntos como veremos más adelante.
2. Generalmente el tipo de dato de cada parámetro deberá ser del mismo tipo. Es decir que si en la llamada a la función el primero es entero y el segundo es real, en la declaración de la función misma el primer parámetro deberá ser entero y el segundo real. En caso contrario se realiza la conversión de tipos.

Ventajas de utilizar funciones con paso de parámetros:

1. Reducen significativamente el tamaño del código.
2. Pueden reutilizarse en otro programa.
3. Se pueden crear librerías o bibliotecas personales que incluyan funciones desarrolladas por el programador.

A continuación mostramos la sintaxis de una función de manera individual.

Pseudocódigo	Lenguaje C
tipo_dato Identif_funcion (parámetros formales)[4]	`tipo_dato Identif_funcion (parámetros formales)`[4]
inicio	{
Declaración de variables	`Declaración de variables;`
Cuerpo de la función	`Cuerpo de la función;`
…………	………………
…………	………………
regresa (valor)[5]	`return (valor);`[5]
Fin	}

Donde:

tipo_dato es el tipo de dato de la función, es decir el valor que regresa la función *identif_funcion* es el nombre de la función.
Parámetros. Son los llamados *formales*, contienen una lista de variables con su tipo y separadas por comas. Éstos son los datos que requiere la función para realizar el cálculo.

Sintaxis de llamadas a una función con paso de parámetros:

Pseudocódigo	Lenguaje C
imprimir Identif_funcion(parametros actuales)[6]	`printf ("%d = ",Identif_funcion(parametros actuales))`[6]`;`
Var ← Identif_funcion (parametros actuales)[6]	`Var = Identif_funcion(parametros actuales)`[6]`;`
si (Identif_funcion(parametros actuales[6])>Exp)	`if (Identif_funcion(parametros actuales)`[6] `>Exp);`

Donde:

Var especifica la variable en donde se guardará el valor devuelto por la función.

A continuación veremos la sintaxis de las funciones sin prototipo.

[4] Cada parámetro debe tener su tipo y nombre; si son varios se separan con una coma.
[5] Es opcional que el valor vaya entre paréntesis.
[6] Nombre de cada parámetro; si son varios se separan con una coma.

Sintaxis funciones con parámetros y con prototipos

Pseudocódigo	Lenguaje C
	# librerías
Variables globales	Variables globales
Constantes	Constantes
tipo_dato funcion_a (parámetros formales)	tipo_dato funcion_a (parámetros formales);
tipo_dato funcion_b (parámetros formales)	tipo_dato funcion_b (parámetros formales);
principal ( )	main( )
inicio	{
Variables locales	Variables locales;
Cuerpo del programa principal	Cuerpo del programa principal
::::::::::::::::::	::::::::::::::::::
c=funcion_a(parámetros actuales)	c=funcion_a(parámetros actuales);
d=funcion_b(parámetros actuales)	d=funcion_b(parámetros actuales);
Instrucción 3	Instrucción 3;
:::::::::::::::::::	:::::::::::::::::::
fin	}
tipo_dato funcion_a (parámetros formales)	tipo_dato funcion_a (parámetros formales)
inicio	{
Variables locales	Variables locales;
Instrucción 1	Instrucción 1;
Instrucción 2	Instrucción 2;
Instrucción 3	Instrucción 3;
regresa (valor)	return (valor);
fin	}
tipo_dato funcion_b (parámetros formales)	tipo_dato funcion_b (parámetros formales)
inicio	{
Variables locales	Variables locales;
Instrucción 1	Instrucción 1;
c=funcion_a(parámetros actuales)	c=funcion_a(parámetros actuales);
Instrucción 2	Instrucción 2;
regresa (valor)	return (valor);
fin	}

Un prototipo de función le da información importante al compilador. En el prototipo se indican el tipo de dato que regresa la función y el número, tipo y orden de parámetros que recibe la misma. El compilador utiliza los prototipos para verificar las llamadas a funciones. Veamos un ejemplo de un prototipo de función:

Ejemplos



```
float promedio(int, float, int); o bien float promedio(int x, float y, int z);
```

Este prototipo indica que la función de nombre máximo retorna como resultado un valor de tipo real. Además informa que la función debe ser llamada con tres parámetros del tipo entero, float y entero. Como podemos apreciar el nombre de los parámetros es opcional.

En ocasiones, será necesario elaborar una función que use un número variable de argumentos como parámetros (en cada llamada). Para ello se utilizan tres puntos en la lista de argumentos de la función. Veamos un ejemplo:

Ejemplos



```
int calcula(int x, ...);
```

El prototipo dice que se requiere como primer parámetro un argumento de tipo int. Después el compilador no comprobará el tipo.

5.6.3 Paso de parámetros en funciones con vectores y matrices

Sintaxis de vectores:

tipo_devuelto NombreFunción(tipo nombre_vector[])

Sintaxis de matrices:

tipo_devuelto NombreFunción(tipo nombre_matriz[] [tam])

Nota 1: Es opcional anotar el tamaño dentro del primer par de paréntesis.
Nota 2: Los arreglos se comportan como parámetros por variable ya que existe un puntero que apunta a la primera posición.

5.7 Ejercicios resueltos de funciones con paso de parámetros

Ejercicio 1. Haga el programa principal y dos funciones con paso de parámetros: 1) sumar dos números enteros y 2) multiplicar dos números enteros.

Pseudocódigo

```
entero suma (entero n1, entero n2)
inicio
    entero s
    s ← n1+n2
    regresa s
fin
entero producto (entero n1, entero n2)
inicio
    entero p
    p ← n1 * n2
    regresa p
fin
principal ( )
inicio
    entero opc, n1, n2, s, p
    imprimir  "¿Qué deseas hacer?"
    imprimir  "1. Suma de dos números"
    imprimir  "2. Producto de dos números"
    leer opc
    si (opc = 1)
    inicio
        imprimir "Dame el primer número entero"; leer n1
        imprimir "Dame el segundo número entero"; leer n2
        s ← suma(n1,n2)
        imprimir "La suma es: " , s
    fin
    si no
    inicio
        imprimir "Dame el primer número entero"; leer n1
        imprimir "Dame el segundo número entero "; leer n2
        p ← producto(n1,n2)
        imprimir "La multiplicación es: " , p
    fin
fin
```

Este programa tiene dos funciones: *entero suma (entero n1, entero n2) y entero producto (entero n1, entero n2).* El encabezado de cada función tiene el tipo de dato que regresa *entero*, el identificador o nombre de la función *suma* o *producto* y los parámetros *(entero n1, entero n2)*. En el cuerpo de la función se declaran las variables locales, en el caso de la suma es sólo una, *entero s*; en seguida encontramos la instrucción que realiza la formula $s \leftarrow n1+n2$ y por último la instrucción que devuelve el resultado al programa que lo llamó *regresa s*. La función producto tiene la misma estructura que la función suma.

El programa principal declara y reserva espacio para cinco variables de tipo entero, luego imprime *¿Qué deseas hacer?* y se muestran dos opciones (1. Suma de dos números, y 2. Producto de dos números); el usuario elige una opción (1 o 2) y el resultado se almacena en la variable *opc*.

Si eligió opc = 1, se piden los datos *Dame el primer número entero y Dame el segundo número entero*; estos dos valores se guardan en las variables *n1* y *n2* y se llama a la función *suma* enviándole los valores de n1 y n2, *s ← suma (n1 , n2);* la función *suma* recibe los valores en su parámetros *n1* y *n2* respectivamente, realiza la fórmula y regresa el resultado almacenado en la variable *s* de la función y lo recibe la variable *s* del programa principal, para posteriormente imprimir el resultado *"La suma es: "*.

Si eligió *opc = 2* llama a la función *producto*, de la misma forma en que se hizo referencia a la función *suma*. Vale la pena mencionar que el programa principal solamente llama a una función cada vez que se corre el mismo.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int suma (int n1, int n2)
{
     int s;
     s = n1+n2;
     return s;
}
int producto(int n1, int n2)
{
     int p;
     p = n1 * n2;
     return (p);
}
main( )
{
     int opc, n1, n2, s, p;
     printf ("¿Que deseas hacer? \n");
     printf ("1. Suma de dos números \n 2. Producto de dos números \n");
     scanf ("%d",&opc);
     if ( opc == 1 )
     {
         printf ("Dame el primer número entero :"); scanf ("%d",&n1);
         printf ("Dame el segundo número entero:"); scanf ("%d",&n2);
         s = suma(n1,n2);
         printf ("La suma es: %d", s);
     }
     else
     {
         printf ("Dame el primer número entero:"); scanf ("%d",&n1);
         printf ("Dame el segundo número entero:"); scanf ("%d",&n2);
```

Una posible salida en pantalla, después de ejecutar el programa sería:

¿Qué deseas hacer?
1. Suma de dos números
2. Producto de dos números
2
Dame el primer número entero: 5
Dame el segundo número entero: 4
La multiplicación es: 20

(*continúa*)

(*continuación*)

Lenguaje C

```
        p = producto(n1,n2);
        printf ("La multiplicación es: %d", p);
    }
    getch(); return 0;
}
```

Otra manera de realizar el ejercicio 1 con menor número de variables locales y menor número de líneas de código lo vemos a continuación.

Pseudocódigo

```
entero suma (entero n1, entero n2)
inicio
    regresa (n1+n2)
fin
entero producto (entero n1, entero n2)
inicio
    regresa (n1 * n2)
fin
principal ( )
inicio
    entero opc, n1, n2
    imprimir "¿Qué deseas hacer?"
    imprimir "1. Suma de dos números "
    imprimir "2. Producto de dos números "
    leer opc
    imprimir "Dame el primer número entero"
    leer n1
    imprimir "Dame el segundo número entero"
    leer n2
    si (opc = 1)
        imprimir "La suma es:", suma(n1, n2)
    si no
        imprimir "La multiplicación es:", producto(n1, n2)
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int suma (int n1, int n2)
{
   return (n1+n2);
} int producto (int n1, int n2)
{
   return (n1 * n2);
}
```

(*continúa*)

(*continuación*)

Lenguaje C

```
main( )
{
   int opc, n1, n2;
   printf ("¿Qué deseas hacer?\n");
   printf ("1. Suma de dos números\n");
   printf ("2. Producto de dos números\n");
   scanf ("%d",&opc);
   printf ("Dame el primer número entero\n");
   scanf ("%d",&n1);
   printf ("Dame el segundo número entero\n");
   scanf ("%d",&n2);
   if (opc == 1)
      printf ("La suma es: %d" , suma(n1,n2));
   else
      printf ("La multiplicación es: %d" , producto(n1,n2));
   getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

¿Qué deseas hacer?
1. Suma de dos números
2. Producto de dos números
1
Dame el primer número entero: 5
Dame el segundo número entero: 4
La suma es: 9

No se requiere declarar en cada función la variable local, ya que en la instrucción *regresa* se agrega de manera directa la fórmula; de esta forma primero se realiza el cálculo y el valor se devuelve al programa que lo llamó. En el programa principal no declaran a *s* y *p* donde se almacenaba el valor que cada función regresaba para posteriormente imprimir este valor. En este caso la instrucción *imprimir* llama de manera directa a la función suma o producto, imprimiendo el resultado que devuelve la función respectiva.

Ejercicio 2. Haga el programa principal y opciones a tres funciones con paso de parámetros que calculen: *a*) la hipotenusa de un triángulo rectángulo, *b*) el área de un triángulo, *c*) el tercer ángulo de un triángulo.

Pseudocódigo

```
 real hipotenusa (entero co, entero ca)
inicio
    real h
    h ← raizcuad (co*co+ ca*ca)
    regresa h
fin
real area (real base, real alt)
inicio
    real a
    a ← base * alt/2
    regresa a
fin
real angulo (real a1, real a2)
inicio
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
        real a3
        a3 ← 180 − ( a1+a2)
        regresa a3
fin
principal ( )
inicio
        caracter opc
        entero ca, co
        real base, alt , h, a, a1, a2, a3
        imprimir   "a) hipotenusa de un triángulo rectángulo "
        imprimir   "b) área de un triángulo "
        Imprimir   "c) tercer ángulo de un triángulo"
        imprimir   "Elige una opción"
        leer opc
        segun_sea (opc)
        inicio
                caso 'a' : imprimir"Escriba la longitud del cateto adyacente al ángulo:"
                           leer ca
                           imprimir "Escriba la longitud del cateto opuesto al ángulo: "
                           leer co
                           h ← hipotenusa (co,ca)
                           imprimir "La hipotenusa del triángulo mide: ", h
                           salir
                caso 'b' : imprimir"Escriba la base y la altura:"
                           leer base, alt
                           a ← area (base,alt)
                           imprimir "El área del triángulo es: ", a
                           salir
                caso 'c' : imprimir"Escriba el valor de los dos ángulos:"
                           leer a1, a2
                           a3 ← angulo (a1,a2)
                           imprimir "El tercer ángulo mide: ", a3
                           salir
        fin
fin
```

Lenguaje C

```c
#include <stdio.h>
#include <conio.h>
#include <math.h>
float hipotenusa (int co, int ca)
{
    float h;
    h = sqrt (co * co + ca * ca);
    return h;
```

(*continúa*)

(*continuación*)

Lenguaje C

```
}
float area (float base, float alt)
{
    float a;
    a = base * alt/2;
    return a;
}
float angulo (float a1, float a2)
{
    float a3;
    a3 = 180 — ( a1+a2);
    return a3;
}
main( )
{
    char opc; int ca, co;
    float base, alt, h, a, a1, a2, a3;
    printf (“a) hipotenusa de un triángulo rectángulo\n”);
    printf (“b) área de un triángulo\n”);
    printf (“c) tercer ángulo de un triángulo\n”);
    printf (“Elige una opción :”);
    scanf (“%c”,&opc);
    switch (opc)
    {
        case ‘a‘ : printf (”Escriba la longitud del cateto adyacente al ángulo: ”);
                   scanf (“%d”,&ca);
                   printf (“Escriba la longitud del cateto opuesto al ángulo: “);
                   scanf (“%d”,&co);
                   h = hipotenusa(co,ca);
                   printf (“La hipotenusa del triángulo mide: %f“, h);
                   break;
        case ‘b’ : printf (”Escriba la base y la altura : ”);
                   scanf (“%f%f”,&base,&alt);
                   a = area (base,alt);
                   printf (“El área del triángulo es: %f“, a);
                   break;
        case ‘c’ : printf (”Escriba el valor de los dos ángulos: ”);
                   scanf (“%f%f”,&a1,&a2);
                   a3 = angulo (a1,a2);
                   printf (“El tercer ángulo mide: %f“, a3);
                   break;
    }
    getch();
    return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

a) hipotenusa de un triángulo rectángulo

b) área de un triángulo.

c) tercer ángulo de un triángulo.

Elige una opción: b.

Escriba la base y la altura: 5.

5

El área del triángulo es: 12.500000.

El ejercicio 2 tiene 3 funciones, cada una con parámetros diferentes (los que se requieren para resolver el problema). El programa principal declara diez variables, una de tipo carácter para seleccionar la opción, dos de tipo entero y siete de tipo real. Si el usuario elige la *opc=c* el programa le solicita que capture el valor de dos ángulos almacenándolos en las variables *a1* y *a2*, posteriormente llama a la función angulo y le envía estos dos valores y espera a que el resultado se almacene en la variable *a3*. La función también tiene dos parámetros de tipo real esperando por los valores *a1*, y *a2*, los cuales reciben estos valores, los procesa y regresa un resultado, el cual será almacenado en la variable *a3*, para posteriormente imprimir este último.

Ejemplos



Si opc = c, a1 =56 y *a2 = 72* se imprimirá *El tercer ángulo mide 52*.

Nota: No es necesario que los parámetros que se envían y los que se encuentran en la función tengan el mismo identificador, lo importante es que sean del mismo tipo.

Este programa también se puede hacer reduciendo variables en el programa principal y líneas de código.

Pseudocódigo

```
real area (real base, real alt)
real hipotenusa (entero co, entero ca)
inicio
    regresa (raizcuad (co*co+ca*ca))
fin
real area (real base, real alt)
inicio
    regresa (base * alt / 2)
fin
real angulo (real a1, real a2)
inicio
    regresa (180 − ( a1+a2))
fin
principal ( )
inicio
    caracter opc
    entero x, y
    real r, b, h, a1, a2
    imprimir  "a) hipotenusa de un triángulo rectángulo "
    imprimir  "b) área de un triángulo "
    Imprimir  "c) tercer ángulo de un triángulo"
    imprimir  "Elige una opción"
    leer opc
    segun_sea (opc)
    inicio
        caso 'a' : imprimir "Escriba la longitud del cateto adyacente al ángulo:"
                   leer x
                   imprimir "Escriba la longitud del cateto opuesto al ángulo:"
                   leer y
                   r ← hipotenusa(x,y)
                   salir
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
            caso ‘b’ : imprimir ”Escriba la base y la altura:”
                       leer b , h
                       r ← area (b,h)
                       salir
            caso ‘c’ : imprimir ”Escriba el valor de los dos ángulos:”
                       leer a1, a2
                       r ← angulo (a1,a2)
                       salir
      fin
      imprimir “El resultado es : “, r
fin
```

En el ejemplo anterior se puede observar que no se utilizan variables locales en las primeras tres funciones y el programa principal usa un menor número de variables. Si el usuario elige la opción *b*, el programa solicita la base y la altura y se almacenan en *b* y *h*; luego se llama a la función *area* y se envían estos dos valores, los cuales son recibidos por los parámetros *base* y *alt* respectivamente. Lo importante es que coincidan en el tipo de dato de la variable *b*, que es real, el parámetro *base* que recibe el valor también es real, lo mismo que la variable *h* y, el parámetro *alt* que recibe el valor también. El valor de *base* y *alt* se utilizan en la función *area* y se genera un valor resultante de tipo *real* que se regresa al programa principal y se almacena en la variable *r*. Independientemente de qué opción haya seleccionado el usuario, todas las funciones regresan un valor de tipo real que es almacenado en la variable *r*.

Ejemplos



Si opc= b, b= 6 h=8, se imprimirá *El resultado es 24.*

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float area (float base, float alt);
float hipotenusa (int co, int ca)
{
   return (sqrt (co * co + ca * ca));
}
float angulo (float a1, float a2)
{
   return (180 – ( a1+a2));
}
main( )
{
   char opc;
   int x, y;
   float r, b, h, a1, a2;
   printf (“a) hipotenusa de un triángulo rectángulo\n”);
   printf (“b) área de un triángulo\n”);
   printf (“c) tercer ángulo de un triángulo\n“);
   printf (“Elige una opción”);
```

Una posible salida en pantalla, después de ejecutar alguno de los programas sería:

a) hipotenusa de un triángulo rectángulo

b) área de un triángulo

c) tercer ángulo de un triángulo

Elige una opción a

Escriba la longitud del cateto adyacente al ángulo: 10

Escriba la longitud del cateto opuesto al ángulo: 7

El resultado es: 12.206555

(*continúa*)

(*continuación*)

Lenguaje C

```
    scanf ("%c",&opc);
    switch (opc)
    {
       case 'a' : printf ("Escriba la longitud del cateto adyacente al ángulo:");
                  scanf ("%d",&x);
                  printf ("Escriba la longitud del cateto opuesto al ángulo:");
                  scanf ("%d",&y);
                  r = hipotenusa(x,y);
                  break;
       case 'b' : printf ("Escriba la base y la altura:");
                  scanf ("%f%f",&b,&h);
                  r = area (b,h);
                  break;
       case 'c' : printf ("Escriba el valor de los dos ángulos:");
                  scanf ("%f%f",&a1,&a2);
                  r = angulo (a1,a2);
                  break;
    }
    printf ("El resultado es : %f", r); getch(); return 0;
}
float area (float base, float alt)
{
    return (base * alt/2);
}
```

En este ejemplo la función que tiene prototipo es area, que como vemos es el mismo encabezado de la declaración de la función, sólo que como hemos comentado en lenguaje C lleva al final punto y coma. Por lo tanto la función fue declarada al final.

Ejercicio 3. Haga el programa principal con opciones a cuatro funciones con paso de parámetros que conviertan de: *a*) grados centígrados a grados Fahrenheit, *b*) grados Fahrenheit a grados centígrados, *c*) grados centígrados a kelvins, *d*) Kelvins a grados centígrados.

Pseudocódigo

```
real convertir_C_F (real c)
inicio
    regresa 1.8 *c +32
fin
real convertir_F_C (real f)
inicio
    regresa ( f −32 ) * 5 / 9
fin
real convertir_C_K (real c)
inicio
    regresa c + 273
fin
real convertir_K_C (real k)
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
inicio
    regresa 273 − k
fin
principal ( )
inicio
    caracter opc
    real x, y
    hacer
    inicio
        imprimir  "Elige una de las siguientes conversiones"
        imprimir  "a) Grados centígrados a grados Fahrenheit"
        imprimir  "b) Grados Fahrenheit a grados centígrados"
        Imprimir  "c) Grados centígrados a kelvins"
        Imprimir  "d) Kelvins a grados centígrados"
        imprimir  "Elige una opción"; leer opc
        imprimir  "Dame los grados que quieres convertir:"; leer x
        segun_sea (opc)
        inicio
            caso 'a' : y = convertir_C_F(x)
                  imprimir "Los grados Fahrenheit son:", y
                  salir
            caso 'b' : y = convertir_F_C(x)
                  imprimir "Los grados centígrados son:", y
                  salir
            caso 'c' : imprimir " Los kelvins son:", convertir_C_K(x)
                  salir
            caso 'd' : imprimir " Los grados centígrados son:", convertir_K_C(x)
                  salir
            en caso contrario; imprimir "Esa opción no existe"
        fin
        imprimir" ¿Deseas repetir el programa S/N? "
        leer opc
    fin
    mientras ( opc = 'S')
fin
```

Este ejercicio utiliza cuatro funciones, una para cada conversión; el programa principal declara tres variables, una de tipo carácter para seleccionar una opción y dos de tipo real para realizar los cálculos. Primero se imprimen en pantalla las cuatro opciones y el usuario elige una opción y se almacena en *opc*, posteriormente solicita los grados que quiere convertir y se almacena en la variable *x*; para cualquier opción la variable *x* tendrá el valor a enviar independientemente de la función y parámetro que reciba.

Ejemplos



En la función *convertir_C_F* el parámetro que recibe es *c*; en la función *convertir_F_C* el parámetro que recibe es *f*; en la función *convertir_C_K* el parámetro que recibe es *c*, y en la función *convertir_K_C* el parámetro que recibe es *k*; todos estos parámetros son de tipo real al igual que la variable *x*, aunque tengan identificadores diferentes.

Ejemplos



Si elige la opc = c, Dame los grados que quieres convertir x = 49, el programa imprimirá *Los grados centígrados son 322. ¿Deseas repetir el programa S/N?* opc = S, volverá a desplegar el menú, con opción a realizar otra conversión.

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
float convertir_C_F (float c)
{
   return 1.8*c +32;
}float convertir_F_C (float f)
{
   return ( f − 32 ) * 5/9 ;
}
float convertir_C_K (float c)
{
   return c + 273;
}
float convertir_K_C (float k )
{
   return 273 − k;
}
main ( )
{
    char opc;
    float x, y;
    do
    {
       printf ("Elige una de las siguientes conversiones\n");
       printf ("a) grados centígrados a grados Fahrenheit\n");
       printf ("b) grados Fahrenheit a grados centígrados\n");
       printf ("c) grados centígrados a kelvins\n");
       printf ("d) kelvins a grados centígrados\n");
       printf ("Elige una opción "); scanf ("%s",&opc);
       printf ("Dame los grados que quieres convertir :");
       scanf ("%f",&x);
       switch (opc)
       {
          case 'a' : y= convertir_C_F(x);
                     printf ("Los grados Fahrenheit son: %f",y);
                     break;
          case 'b' : y= convertir_F_C(x);
                     printf ("Los grados centígrados son: %f",y);
                     break;
          case 'c' : printf ("Los kelvins son :%f" ,convertir_C_K(x));
                     break;
          case 'd' : printf ("Los grados centígrados son : %f",convertir_K_C(x));
                     break;
          default: printf ("Esa opción no existe");
       }
       printf ("¿Deseas repetir el programa S/N? "); scanf ("%s",&opc);
    } while ( opc == 'S'); return 0;
}
```

Una posible salida en pantalla, después de ejecutar alguno de los programas sería:

Elige una de las siguientes conversiones:

a) Grados centígrados a grados Fahrenheit.
b) Grados Fahrenheit a grados centígrados.
c) Grados centígrados a kelvins.
d) Kelvins a grados centígrados.

Elige una opción a

Dame los grados que quieres convertir: 45.

Los grados Fahrenheit son: 113.000000 ¿Deseas repetir el programa S/N? S.

Elige una de las siguientes conversiones:

a) Grados centígrados a grados Fahrenheit.

b) Grados Fahrenheit a grados centígrados.

c) Grados centígrados a kelvins.

d) Kelvins a grados centígrados.

Elige una opción c

Dame los grados que quieres convertir: 45.

Los kelvins son: 318.000000 ¿Deseas repetir el programa S/N? N.

Ejercicio 4. Haga el programa principal con opciones a cinco funciones con paso de parámetros que calcule las siguientes áreas: 1) círculo, 2) cuadrado, 3) rectángulo, 4) triángulo, 5) trapecio.

Pseudocódigo

```
constante PI ← 3.1416
real area_circulo (entero radio)
inicio
     real area_cir
     area_cir ← PI * radio * radio
     regresa area_cir
fin
entero area_cuadrado (entero lado)
inicio
     entero area_cua
     area_cua ← lado * lado
     regresa area_cua
fin
entero area_rectangulo (entero base, entero alt)
inicio
     entero area_rect
     area_rect ← base * alt
     regresa area_rect
fin
real area_triangulo (real base, real alt)
inicio
     real area_tria
     area_tria ← base * alt / 2
     regresa area_tria
fin
real area_trapecio (real basemay, real basemen, real alt)
inicio
     real area_tra
     area_tra ← (basemay + basemen) * alt / 2
     regresa area_tra
fin
principal ( )
inicio
     entero opc, a, b, c; real d, x, y, z
     hacer
     inicio
          imprimir "Elige una de las siguientes áreas"
          imprimir "1)Círculo 2)Cuadrado 3)Rectángulo 4)Triángulo 5)Trapecio"
          leer opc
          segun_sea (opc)
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
        inicio
            caso 1 : imprimir"Escriba el radio:" ; leer a
                  d←area_circulo (a); imprimir "El área del circulo es ", d
                  salir
            caso 2 : imprimir"Escriba el lado:" ; leer a
                  b←area_cuadrado(a); imprimir "El área del cuadrado es ", b
                  salir
            caso 3 : imprimir"Escriba la base y la altura:" ; leer a, b
                  c← area_rectangulo(a,b); imprimir "El área del rectángulo es ", c
                  salir
            caso 4 : imprimir"Escriba la base y la altura:" ; leer x, y
                  d←area_triangulo(x,y); imprimir "El área del triángulo es ", d
                  salir
            caso 5 : imprimir"Escriba la base mayor, la base menor y la altura:"; leer x , y , z
                  d←area_trapecio(x,y,z); imprimir "El área del trapecio es ", d
                  salir
        fin
        imprimir" ¿Deseas repetir el programa S=1 / N=2? "; leer opc
    fin
    mientras (opc = 1)
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#define PI 3.1416
float area_circulo (int radio );
int area_cuadrado (int lado);
int area_rectangulo (int base, int alt)
{
   int area_rect;
   area_rect = base * alt;
   return area_rect;
}
float area_triangulo (float base, float alt)
{
   float area_tria;
   area_tria = base * alt/2;
   return area_tria;
}
float area_trapecio (float basemay, float basemen, float alt)
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Elige una de las siguientes áreas

1) Círculo.

2) Cuadrado.

3) Rectángulo.

4) Triángulo.

5) Trapecio.

5

Escriba la base mayor la base menor y la altura: 4

7, 6

El área del trapecio es 33.000000 Deseas repetir el programa S51 / N 2:2

(*continúa*)

(*continuación*)

Lenguaje C

```
{
   float area_tra;
   area_tra = (basemay +basemen)* alt/2;
   return area_tra;
}
main( )
{
   int opc, a, b, c; float d,x,y,z;
do
   {
        printf ("Elige una de las siguientes áreas\n");
        printf ("1)Círculo\n");
        printf ("2)Cuadrado\n");
        printf ("3)Rectángulo\n");
        printf ("4)Triángulo\n");
        printf ("5)Trapecio\n");
        scanf ("%d",&opc);
        switch (opc)
        {
           case 1 : printf ("Escriba el radio :" );
                    scanf ("%d",&a);
                    d=area_circulo (a);
                    printf ( "El área del circulo es %f", d);
                    break;
           case 2 : printf ("Escriba el lado :" );
                    scanf ("%d",&a);
                    b=area_cuadrado(a);
                    printf ( "El área del cuadrado es %d", b);
                    break;
           case 3 : printf ("Escriba la base y la altura :");
                    scanf ("%d%d",&a,&b);
                    c=area_rectangulo(a,b);
                    printf ( "El área del rectángulo es %d", c);
                    break;
           case 4 : printf ("Escriba la base y la altura :");
                    scanf ("%f%f",&x,&y);
                    d=area_triangulo(x,y);
                    printf ( "El área del triángulo es %f", d);
                    break;
           case 5 : printf ("Escriba la base mayor la base menor y la altura:");
                    scanf ("%f%f%f",&x,&y,&z);
                    d=area_trapecio(x,y,z);
```

(*continúa*)

(*continuación*)

Lenguaje C

```
                    printf ( "El área del trapecio es %f", d);
                    break;
          } printf ("¿Deseas repetir el programa S=1 / N=2? "); scanf ("%d",&opc);
    } while ( opc == 1); return 0;
}
float area_circulo (int radio)
{
    float area_cir;
    area_cir = PI * radio * radio;
    return area_cir;
}
int area_cuadrado (int lado)
{
    int area_cua;
    area_cua = lado * lado;
    return area_cua;
}
```

Este ejercicio tiene 5 funciones, cada una con parámetros diferentes (los que requieren para resolver el problema). El programa principal declara ocho variables, cuatro de tipo entero y cuatro de tipo real. Si el usuario elige la *opc = 5* el programa le solicita que capture el valor de la base mayor, la base menor y la altura almacenándolos en las variables *x, y* y *z*, las tres son de tipo real. Posteriormente llama a la función *area_trapecio* y le envían estos tres valores, los cuales son recibidos por los parámetros *basemay, baremen* y *alt* (declarados de tipo real), la función regresa un valor real el cual se almacena en la variable *d* del programa principal, que también es de tipo real. Para imprimir el valor de la variable *d*.

Ejemplos



Si opc = 5, x = 6, y = 4, z = 5 imprimirá *El área del trapecio es 25. ¿Deseas repetir el programa S = 1 / N = 2? opc = 1, volverá a desplegar el menú, con opción a realizar el cálculo de un área; si opc = 2 termina el programa*.

Nota: El programa principal sólo podrá imprimir las variables que hayan sido declaradas en el mismo; es decir no podrá imprimir parámetros o variables locales declaradas en las funciones. En lenguaje C a diferencia del pseudocódigo utilizamos prototipos para las funciones *area_circulo* y *area_cuadrado*.

Ejercicio 5. Realice tres funciones con paso de parámetros para realizar: 1) área de un círculo, 2) área de un aro, 3) volumen de un cilindro, utilizando la función área del círculo para resolver los incisos 2) y 3). Realice el programa principal con las tres opciones anteriores.

Este ejercicio tiene 3 funciones, cada una con sus diferentes parámetros. El programa principal declara cinco variables, una de tipo entero para seleccionar la opción y cuatro de tipo real para resolver cualquiera de las funciones. Si el usuario elige la *opc = 2* el programa le solicita que capture el valor del radio mayor y del radio menor almacenándolos en las variables *radio1* y *radio2*, las dos de tipo real; posteriormente llama a la función *area_aro* y le envían los valores de *radio1* y *radio2*, los cuales son recibidos por los parámetros *r1* y *r2*. Hay que recordar que radio1 deberá ser del mismo tipo que *r1* y radio2 deberá ser del mismo tipo que *r2*. Esta función llama dos veces a la función *area_circulo*, pero con dos radios diferentes, una vez para calcular el área del círculo mayor con *r1* y la segunda vez para calcular el área del circulo menor con *r2* y cuando se tienen los dos valores que regresa la función se realiza la diferencia para tener el área del aro; esta función regresa un valor real el cual se almacena en la variable *res*.

Ejemplos



Si *opc =2, radio1= 7, radio2= 4*, imprimirá *El área del aro es* 103.66, ya que la función *area_circulo (7)* regresa 153.93 y *area_circulo (4)* regresa 50.26, la diferencia 153.93−50.26 = 103.66.

Nota: Una función puede ser llamada desde el programa principal o desde otra función.

Pseudocódigo

```
constante pi ← 3.1416
real area_circulo (real r)
inicio
    real area_cir
    area_cir ← pi * r * r
    regresa area_cir
fin
real area_aro (real r1, real r2)
inicio
    real a_aro
    a_aro ← area_circulo(r1) − area_circulo(r2)
    regresa a_aro
fin
real volumen_cilindro(real r, real h)
inicio
    real v_cil
    v_cil ← area_circulo( r) * h
    regresa v_cil
fin
principal ( )
inicio
entero opc
real radio1, radio2, altura, res
    hacer
    inicio
        imprimir    "Elige una de las siguientes áreas"
        imprimir    "1) Área del círculo"
        imprimir    "2) Área del aro"
        Imprimir    "3) Volumen del cilindro"
        leer opc
        segun_sea (opc)
        inicio
            caso 1 : imprimir "Escriba el radio:"
                     leer radio1
                     res←area_circulo (radio1)
                     imprimir "El área del círculo es", res
                     salir
            caso 2 : imprimir "Escriba el radio mayor y el radio menor:";
                     leer radio1, radio2
                     res←area_aro(radio1, radio2)
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
                                imprimir "El área del aro es: ", res
                                salir
                     caso 3 : imprimir"Escriba el radio y la altura:"
                                leer radio1, altura
                                res← volumen_cilindro(radio1,altura)
                                imprimir "El volumen del cilindro es: ", res
              fin
              imprimir"¿Deseas repetir el programa S=1 / N=2? ";
              leer opc
       fin
       mientras ( opc = 1)
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#define PI 3.1416
float area_circulo (float r)
{
   float area_cir;
   area_cir = PI * r * r;
   return area_cir;
}
float area_aro (float r1, float r2)
{
   float a_aro ;
   a_aro = area_circulo(r1)—area_circulo (r2);
   return a_aro;
}
float volumen_cilindro(float r, float h)
{
   float v_cil;
   v_cil = area_circulo(r) * h;
   return v_cil;
}
main( )
{
   int opc;
   float radio1, radio2, altura, res;
   do
   {
        printf ("Elige una de las siguientes áreas \n");
        printf ("1)Área del círculo\n");
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Elige una de las siguientes áreas

1) Área del círculo.

2) Área del aro.

3) Volumen del cilindro.

3

Escriba el radio y la altura: 7. 4

El volumen del cilindro es: 615.753601.

¿Deseas repetir el programa S=1 / N=2?
1

Elige una de las siguientes áreas

1) Área del círculo.

2) Área del aro.

3) Volumen del cilindro.

2

Escribe el radio mayor y el radio menor:
4 2

El área del aro es: 37.699200.

¿Deseas repetir el programa S=1 / N=2?
2

(*continúa*)

(*continuación*)

Lenguaje C

```
        printf (“2)Área del aro\n”);
        printf (“3)Volumen del cilindro\n”);
        scanf (“%d”,&opc);
        switch (opc)
        {
             case 1 : printf (”Escriba el radio:” );
                      scanf (“%f”,&radio1);
                      res=area_circulo (radio1);
                      printf (“El área del circulo es %f“, res);
                      break;
             case 2 : printf (”Escriba el radio mayor y el radio menor:”);
                      scanf (“%f%f”,&radio1,&radio2);
                      res=area_aro(radio1, radio2);
                      printf (“El área del aro es : %f“,res);
                      break;
             case 3 : printf (“Escriba el radio y la altura:”);
                      scanf (“%f%f”,&radio1,&altura);
                      res= volumen_cilindro(radio1, altura);
                      printf (“El volumen del cilindro es: %f“, res);
                      break;
        }
        printf (“\n¿Deseas repetir el programa S=1 / N=2? ”);
        scanf (“%d”,&opc);
    }while ( opc == 1);
    return 0;
}
```

Ejercicio 6. Sume dos vectores de 20 elementos como máximo y almacenar el resultado en un tercer vector utilizando funciones con paso de parámetros.

Pseudocódigo	Lenguaje C
	#include<stdio.h>
	#include<conio.h>
llenar_vector(entero x[20], entero tam)	void llenar_vector(int x[20],int tam)
inicio	{
entero i	int i;
desde (i←0, i< tam, i←i+1)	for (i=0; i < tam; i++)
leer x[i]	scanf (“%d”,&x[i]);
fin	}
nada imprimir_vector(entero x[ ], entero tam)	void imprimir_vector(int x[], int tam)
inicio	{
entero i	int i;
desde (i←0, i< tam, i←i+1)	for (i=0; i< tam; i++)

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
imprimir x[i]	`printf("%d ",x[i]);`
fin	`}`
nada sum_v(entero x[20], entero y[ ], entero z[20], entero t)	`void sum_v(int x[20],int y[],int  z[20],int t)`
inicio	`{`
entero i	`int i;`
desde (i←0, i< t, i←i+1)	`for (i=0; i<t; i++)`
z[i] ← x[i] +y[i]	`z[i]=x[i]+y[i];`
fin	`}`
principal()	`main()`
inicio	`{`
entero t, a[20], b[20], c[20]	`int t, a[20], b[20], c[20];`
imprimir "Total de elementos: "	`printf("Total de elementos: ");`
leer t	`scanf ("%d",&t);`
imprimir "Dame los elementos del vector a: "	`printf("Dame los elementos del vector a:\n");`
llenar_vector(a,t)	`llenar_vector(a,t);`
imprimir "Dame los elementos del vector b: "	`printf("Dame los elementos del vector b:\n");`
llenar_vector(b,t)	`llenar_vector(b,t);`
sum_v(a,b,c,t)	`sum_v(a,b,c,t);`
imprimir "El vector a es: "	`printf("El vector a es: ");`
imprimir_vector(a,t)	`imprimir_vector(a,t);`
imprimir "El vector b es: "	`printf("\nEl vector b es: ");`
imprimir_vector(b,t)	`imprimir_vector(b,t);`
imprimir "La suma es: "	`printf("\nLa suma es: ");`
imprimir_vector(c,t)	`imprimir_vector(c,t);`
	`getch(); return 0;`
fin	`}`

Este ejercicio tiene tres funciones que son utilizadas por el programa principal más de una vez. En el programa principal (main) se declaran tres arreglos de 20 elementos cada uno, denominados *a,b,c* y una variable *t* que almacena el total de elementos, se llama a la función *llenar_vector(a,t)* mandándole *a* y *t*, la función recibe en *x* a la *a* y en *tam* a la *t*, la función realiza el ciclo *desde* y al terminar todo lo que se almacenó en la variable *x* se le pasa a la variable *a*. Cabe recordar que los parámetros que pasan arreglos se comportan como parámetros por *valor* (los datos salen modificados). Se llama a la función *llenar_vector* tres veces, almacenando los datos en los arreglos *a* y *b*. Posteriormente se llama a la función *sum_ v(a,b,c,t)* en el cual el arreglo *a* y *b* ya contienen datos y el arreglo *c* se utilizará para almacenar la suma de los vectores *a* y *b*, *t* envía el tamaño del vector el cual es recibido por el parámetro *tam*, igual que en las otras funciones. Al finalizar la función *sum_v(a,b,c,t)* los valores almacenados en el arreglo z, pasarán al arreglo c. Posteriormente se llama tres veces a la función *imprimir_vector(a,t)* para imprimir los datos de los vectores *a, b,* y *c*.

Ejemplos



Si *t* es igual a 5 y se almacenan en *a*[7,9,3,6,1] y en *b*[2,5,8,3,4], el resultado se almacenará en *c*[9,14,11,9,5].

Nota: En el parámetro *formal* el índice es opcional.

Ejercicio 7. Sume dos matrices cuadradas de máximo 20 elementos como máximo y almacenar el resultado en una tercera matriz utilizando funciones con paso de parámetros.

Pseudocódigo	Lenguaje C
	`#include<stdio.h>`
	`#include<conio.h>`
llenar_matriz(entero x[ ][20], entero tam)	`void llenar_matriz(int x[][20], int tam)`
inicio	`{`
entero i, j	`int i,j;`
desde (i←0, i< tam, i←i+1)	`for (i=0; i< tam; i++)`
desde (j←0, j< tam, j←j+1)	`for (j=0; j< tam; j++)`
leer x[i][j]	`scanf ("%d",&x[i][j]);`
fin	`}`
nada imprimir_matriz(entero x[ ][20], entero tam)	`void imprimir_matriz(int x[][20],int tam)`
inicio	`{`
entero i, j	`int i,j;`
desde (i←0, i<tam, i←i+1)	`for (i=0;i< tam; i++)`
inicio	`{`
desde (j←0, j< tam, j←j+1)	`for (j=0; j< tam; j++)`
imprimir x[i][j]	`printf("%d ",x[i][j]);`
salto_linea	`printf("\n");`
fin	`}`
fin	`}`
nada sum(entero x[ ][20],entero y[ ][20], entero z[ ][20],entero t)	`void sum(int x[][20],int y[][20],int z[][20],int t)`
inicio	`{`
entero i, j	`int i,j;`
desde (i←0, i< t, i←i+1)	`for (i=0; i< t ; i++)`
desde (j←0, j< t, j←j+1)	`for (j=0; j< t; j++)`
z[i][j] ← x[i][j] +y[i][j]	`z[i][j]= x[i][j] + y[i][j];`
fin	`}`
principal()	`main()`
inicio	`{`
entero t, a[20] [20], b[20] [20], c[20] [20]	`int t, a[20][20], b[20][20], c [20][20];`
imprimir "Tamaño de la matriz:"; leer t	`printf("Tamaño de la matriz:"); scanf("%d",&t);`
imprimir "Dame los elementos de la matriz a: "	`printf("Dame los elementos de la matriz a:\n");`
llenar_matriz(a,t)	`llenar_matriz(a,t);`
imprimir "Dame los elementos de la matriz b: "	`printf("Dame los elementos de la matriz b:\n");`
llenar_matriz(b,t)	`llenar_matriz(b,t);`
sum(a,b,c,t)	`sum(a,b,c,t);`
imprimir "La matriz a es: "	`printf("La matriz a es:\n");`
imprimir_matriz(a,t)	`imprimir_matriz(a,t);`
imprimir "La matriz b es: "	`printf("\nLa matriz b es:\n");`
imprimir_matriz(b,t)	`imprimir_matriz(b,t);`
imprimir "La suma es: "	`printf("\nLa suma es:\n");`
imprimir_matriz(c,t)	`imprimir_matriz(c,t);`
	`getch(); return 0;`
fin	`}`

Este ejercicio es semejante al 6, con la diferencia de que el 7 suma matrices; por lo tanto necesitamos dos índices y dos ciclos *desde (for)* para el recorrido de las matrices. Se declaran tres arreglos de 400 elementos cada uno, denominados *a,b,c* y una variable *t* (tamaño matriz), se llama a la función *llenar_matriz(a,t)* mandándole *a* y *t*, la función recibe en *x* a la *a* y en *tam* a la *t*, la función realiza el ciclo *desde* y al terminar lo que se almacenó en la variable *x* se le pasa a la variable *a*. Los parámetros que pasan arreglos se comportan como parámetros por *valor* (los datos salen modificados). Se llama a la función *llenar_vector* tres veces, almacenando los datos en los arreglos *a* y *b*. Se llama a la función *sum(a,b,c,t)* en la cual el arreglo *a* y *b* ya contienen datos y en el arreglo *c* se almacena la suma de las matrices *a* y *b*. Al finalizar la función *sum(a,b,c,t)* los valores almacenados en el arreglo z, pasarán al arreglo c. Se llama tres veces a la función *imprimir_ matriz(a,t)* para imprimir los datos de las matrices *a, b,* y *c*.

Nota: En el parámetro formal el primer índice es opcional y el segundo es obligado.

5.8 Recursividad

Cuando una función se llama a sí misma le llamamos *recursividad*. Para finalizar ésta debe existir una condición previamente definida. A continuación presentamos un ejemplo.

Ejemplos



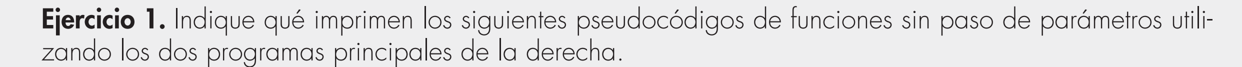
Factorial de un número

Si aplicamos la propiedad asociativa de la multiplicación: n! = n*(n −1)*(n −2)* ... *2*1 y agrupamos los factores: n! = n * [(n −1)*(n −2)* ... *2*1], obtenemos: n! = n * (n −1)!, la recursividad terminará cuando n=0, retornando 1.

Pseudocódigo	Lenguaje C
	`#include <conio.h>`
	`#include <stdio.h>`
real factorial(entero n)	`float factorial(int n)`
inicio	`{`
real f	`float f;`
si (n=0) f←1	`if (n==0) f=1;`
si no f←n*factorial(n−1)	`else f=n*factorial(n−1);`
regresa f	`return f;`
fin	`}`
principal()	`main()`
inicio	`{`
real fact	`float fact;`
entero n	`int n;`
imprimir “Dame un número: “	`printf(“\nDame un número: “);`
leer n	`scanf(“%d”,&n);`
fact←factorial(n)	`fact=factorial(n);`
imprimir “El factorial =”,fact	`printf(“\nEl factorial=%7.2f\n”,fact);`
	`getch(); return 0;`
fin	`}`

En el ejercicio anterior se lee un número entero *n* y la variable *fact* almacenará el resultado de la función recursiva factorial después de que termine. Si por ejemplo *n=4*, entra a la función y la variable *f = 4*factorial(3)=4*3*factorial(2)=4*3*2*factorial(1)*, como *n=1* la condición es verdadera, *f= 4*3*2*1*; la función termina la recursividad y regresa el valor de 24. Al final se imprimirá *El factorial*= 24.00.

Ejercicios complementarios de funciones en pseudocódigo



Ejercicio 1. Indique qué imprimen los siguientes pseudocódigos de funciones sin paso de parámetros utilizando los dos programas principales de la derecha.

```
nada funcion_a ( )
inicio
    imprimir "México"
fin
nada funcion_b ( )
inicio
    imprimir "Italia"
    imprimir "Chile"
fin
nada funcion_c ( )
inicio
    imprimir "Japon"
fin
nada funcion_d ( )
inicio
    imprimir "Senegal"
fin
```

```
principal ( )
inicio
    funcion_b( )
    funcion_a( )
    funcion_d( )
    funcion_c( )
fin
______________________________
______________________________
______________________________
______________________________
______________________________
```

```
principal ( )
inicio
    funcion_a( )
    funcion_d( )
    funcion_b( )
    funcion_c( )
    funcion_a( )
fin
______________________________
______________________________
______________________________
______________________________
______________________________
______________________________
```

Ejercicio 2. Complete el pseudocódigo con tres funciones sin paso de parámetros para calcular el perímetro de *a*) triángulo equilátero, *b*) cuadrado, *c*) rectángulo.

```
nada peri_triangulo ( )
inicio
    real ____________, p
    imprimir "Dame el lado del triángulo:" ; leer lado
    p←__________________________; imprimir "El perímetro del triángulo es " _________
fin
nada peri_cuadrado ( )
inicio
    real lado, p
    imprimir "Dame el lado del cuadrado:"; ____________________
    _________←lado *4; imprimir "El perímetro del cuadrado es " p
fin
nada __________________________ ()
inicio
    real _________, lado2, p
```

(*continúa*)

(*continuación*)

```
    imprimir "Dame la altura y la base del rectángulo:"; _______________ lado1, lado2
    p←_________________________;
    imprimir "El perímetro del rectángulo es ", p
fin
principal ( )
inicio
    entero opc
    imprimir "1)Perímetro del triángulo 2)Perímetro del cuadrado 3)Perímetro del rectángulo"
    leer opc
    si (opc=1) ______________________________
    si no
        si (opc=2) peri_cuadrado( )
        si no _______________________________
fin
```

Ejercicio 3. Complete el siguiente pseudocódigo con funciones sin y con paso de parámetros para calcular el factorial de un número entero y la potencia de un número entero y completar el programa principal.

3a) Funciones sin paso de parámetros	3b) Funciones con paso de parámetros
nada factorial ()	entero factorial (entero _________)
inicio	inicio
entero i, x, fact ←1	entero i,fact ←1
imprimir "Dame el número:"	desde (i← x, i > 1, i ← i −1)
leer x	fact←fact * i
desde (i← x, ___________, i ← i − 1)	regresa ______________
fact←fact * i	fin
imprimir "El factorial de" , x ," = ", ___________	______________ potencia(entero x, entero _________)
fin	inicio
nada potencia()	entero i, ____________←1
inicio	desde (i←1,i <=y, i←i +1)
entero i,x,y,res←1	res←res * x
Imprimir "Escriba el número:"	regresa ______________
leer ___________	fin
imprimir "A qué potencia"	principal ( )
leer ___________	inicio
desde (i←1,i <=y, i←i +1)	
res←res * x;	entero _________ , a,b,r
imprimir "La potencia es", res	imprimir "1) Factorial 2) Elevar a una potencia"; leer opc
fin	si (opc =1)
principal ( )	inicio
inicio	imprimir " Dame el número: "; leer a
entero opc	r ← ________________________

(*continúa*)

(*continuación*)

3a) Funciones sin paso de parámetros	3b) Funciones con paso de parámetros
imprimir "1) Factorial 2) Elevar a una potencia"	fin
leer opc	si no
si (opc =1)	inicio
______________________	imprimir "Escriba el número: "; leer a
si no	imprimir "A qué potencia"; leer b
______________________	r ← ____________________
fin	fin
	imprimir "resultado es" _________
	fin

Ejercicio 4. Complete el siguiente pseudocódigo con tres funciones con paso de parámetros que realizan lo siguiente:

a) Encuentra el mayor entre tres números enteros
b) Calcula el promedio entre un conjunto de *n* números almacenados en un arreglo
c) Ordena el arreglo utilizando el método de burbuja

```
__________mayor (entero a, entero b, _________)
inicio
    si _____________________ regresa a
    si no
        si ___________________ regresa ________
        si no regresa c
fin
_________ promedio (real a[20],_________)
inicio
    entero i; real sum←0
    desde (_____________________)
        sum← ____________
    regresa ___________
fin
nada _________________________________(real a[ ], entero n)
inicio
    entero i, j; real aux
    desde (j←1,_______,j← j+1)
        desde ( ___________, i < n−j, i←i+1)
            si (a[i]> a[i+1])
                inicio
                    aux ← a[i]
                    ____________ ←a[i+1]
                    a[i+1] ← aux
                fin
```

(*continúa*)

(*continuación*)

```
fin
nada principal ()
inicio
    ___________x, y, n, i
    real p, a[20]
    ___________ opc
    imprimir "a) devuelve el mayor de tres números b) calcula el promedio entre un conjunto de números enteros"
    imprimir "c) ordena un arreglo de tamaño variado"
    leer _________
    ________________ (opc)
    inicio
    caso 'a': leer x, y, n
         _______________________
         salir
    caso 'b': leer _________
         desde (i←0, i< n, i←i+1)
              leer a[i]
         p←_____________________(a, n)
         imprimir ___________
         salir
    caso 'c': leer _________
         desde (i←0, i< n, i←i+1)
              leer a[i]
         burbuja(______________)
         imprimir " el arreglo ordenado queda"
         desde (i←0, i< n, i←i+1)
              imprimir _____________
         salir
    fin
fin
```

Ejercicio 5. Sume dos matrices de máximo 10 × 10 elementos y almacenar el resultado en una tercera matriz utilizando funciones con paso de parámetros.

```
nada __________________________( _____________ x[10][10], entero f, entero c)
inicio
    entero i , j
    desde (i←0, i< f, i←i+1)
         desde (j←0, j< ____, j←j+1)
    ______________________
fin
nada imp_matriz (________________, entero f, entero c)
```

(*continúa*)

(continuación)

```
inicio
    entero i, j
    desde (__________________)
        desde (j←0, j<c, j←j+1)
            imprimir x[i][j]
fin
nada sum_matriz (______________ x[10][10], entero y[10][10], ______________ z[10][10],entero f, entero c)
inicio
    entero i , j
    desde (i←0, i<______, i←i+1)
        desde (j←0, j< ________, j←j+1)
            ___________________________________
fin
principal()
inicio
    entero f,c, m1[10][10], m2[10][10], m3[10][10]
    imprimir " introduce el número de filas y columnas";    leer _________________
    llen_matriz(_______________)
    __________________(m2,f,c)
    sum_matriz(__________________________________)
    imprimir "la primera matriz es:"; ____________________________
    imprimir "la segunda matriz es:";  imp_matriz(______________)
    imprimir "la suma de matrices es:"
    ___________________(m3,f,c)
fin
```

Ejercicios complementarios de funciones en lenguaje C

Parte I. Realice una corrida a mano de los siguientes programas

Ejercicio 1

```c
int acum(int vector[], int x)
{
    int i, suma=0;
    for(i=0; i<x; i++)
        suma+=vector[i];
```

Ejercicio 2

```c
void mult(int mat1[][3], int mat2[][3], int vec[], int x)
{
    int i,j;
    for( i=0; i<x; i++)
        for( j=0; j<x; j++)
```

(continúa)

(*continuación*)

```
   return(suma);
}
int suma(int x,int y=3)
{
   return x+y;
}
main( )
{
   int v[6]={2,7,4,3,10},s;
   s=acum(v,suma(3));
   printf("%d",s);
   getch();
}
```

```
      mat2[i][j]=mat1[j][i]*vec[j];
}
int acum(int mat2[][3], int x)
{
      int i,j,acum=0;
      for( i=0; i<x; i++)
          for( j=0; j<x; j++)
              acum+=mat2[i][j];
          return acum;
}
main( )
{
      int v[3]={2,3,4},m1[3][3]= {1, 2, 3, 4, 5, 2, 1, 5},
        m2[3][3], s;
      mult(m1,m2,v,3); s=acum(m2,3);
      printf("%d",s); getch();
}
```

Parte II. Realice una corrida a mano del siguiente programa y conteste las preguntas

```
float n1,n2;
void datos()
{
   printf("dame el 1er. No.");     scanf("%f",&n1);
   printf("dame el 2do. No.");     scanf("%f",&n2);
}
float suma()
{
   float r;
   datos(); r=n1+n2;
   return(r);
}
void resta()
{
   float r; datos();
   r=n1−n2; printf("Resta= %f",r);
}
float multi()
{
   float r;
   datos(); r=n1*n2;
   return(r);
```

(*continúa*)

(*continuación*)

```
}
void divi()
{
   float r;
   datos(); r=n1/n2;
   printf("División= %f",r);
}
main()
{
   int op;
   do
   {printf("operaciones básicas\n1) suma\n2)resta\n3)multiplicacion\n4)division\n5)
     salir\n");
   printf("dame tu opción:");      scanf("%d",&op);
   switch(op)
   {
      case 1:printf("suma = %f",suma()); break;
      case 2:resta(); break;
      case 3:printf("multiplicacion = %f",multi()); break;
      case 4:divi(); break;
   }getch(); }while(op!=5);
}
```

1. ¿Qué es lo que hace el programa?

   ____

2. ¿Qué tipo de variables son *n1* y *n2*?______________________________
3. ¿Qué tipo de variable es *op*?__________________________________
4. ¿Qué diferencia hay entre las funciones *suma* y *resta*?

   ____

5. ¿Las funciones utilizan o *no* parámetros?

   ____

6. ¿Cuál es la desventaja de las variables globales?

   ____

7. ¿Qué pasaría si declaramos cualquier función después del main?

   ____

8. ¿Cuál es la ventaja de utilizar prototipos?

   ____

Elabore el programa anterior con las siguientes características:

a) No utilice variables globales. *b*) Utilice funciones con parámetros.

Parte III. Complete las siguientes líneas y realice la prueba de escritorio o corrida a mano de los siguientes programas:

Ejercicio 1

```
int d;
int Subpro_1 (int m);
void Subpro_2 (int i)
{
   if (Subpro_1(i)>2)printf("%d\n",i*5);
      else        printf ("%d\n",i*3);
}
float Subpro_3 (________, float s)
{
   ________ w;
   if (r>s) w=r/s;
      else w=s/r;
   return w;
}
main()
{
   float a=7, b=5, c;
   c=Subpro_3(___,b);
   printf ("%f\n",c);
   for (d=2;d<9;d+=2)
      Subpro_2(d);
   getch();
}
_____ Subpro_1 (int m)
{
   int w;
   w=m/2;
   return w;
}
```

Ejercicio 2

```
float f2(float m, _______);
______ f3(________)
{
   int i,z=1;
   for(i=1;i<=x;i++)
      z=z*i;
   printf("Valor 2:%d\n ",z);
}
float f1(float a, float b)
{
   float z;
   z=f2(a,b)/4;
   ________(z);
}
main()
{
   float x,y=3; int z=2;
      for (x=2;x<5;x=x+2)
      {
         printf("Valor 1:%f\n",f1(x,y));
         f3(z);
      }
   getch();
}
float f2(float m, ________)
{
      if(m>n) return(m+n);
         else return(m*n);
}
```

Parte IV. Elabore la codificación en lenguaje C de los siguientes programas utilizando funciones por valor con paso de parámetros (realice el cálculo e impresión respectivas):

1. Leer un vector de enteros y un número entero *x* e imprimir la cantidad de:
   a) veces que *x* aparece en el arreglo.
   b) elementos menores a *x* en el arreglo.
   c) elementos mayores a *x* en el arreglo.

2. Leer un arreglo de enteros e imprimir la cantidad de valores del arreglo que son números primos. Utilizar una función que reciba un número entero y que decida si el número es primo.
3. Leer dos números enteros positivos y en una función calcular su producto mediante sumas sucesivas.
4. Para *n* términos leer un número real *x* e imprimir la sumatoria utilizando la siguientes series:

   a) $1!+2!+3!+\cdots+n!$

   b) $\frac{(x-1)^2}{0!}+\frac{(x-3)^4}{2!}+\frac{(x-5)^2}{4!}+\frac{(x-7)^4}{6!}+\cdots$

   c) $e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}\cdots+\frac{x^n}{n!}$

   d) $\text{sen}(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\frac{x^9}{9!}\cdots$

   e) $\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}\cdots$

5. Leer *m* y *n*. Imprimir el valor del número combinatorio: $\binom{m}{n}=\frac{m!}{(m-n)!n!}$

6. Leer *n* datos para un vector de números reales, utilizando funciones para calcular la cantidad de: valores impares, valores pares y veces que se repite el valor máximo del arreglo.
7. Leer un vector de tamaño *n* e imprimir la media aritmética y la semisuma de los valores máximo y mínimo.
8. Imprimir una flecha con asteriscos; utilizar dos funciones, una imprimirá el triángulo y la otra el rectángulo. Preguntar la base del triángulo, la altura y ancho del rectángulo, por ejemplo si bas_triang = 5, h_rect = 4 y anc_rect = 3, se imprimirá:

```
    *
   * *
  * * *
 * * * *
* * * * *
  * * *
  * * *
  * * *
  * * *
```

9. Imprimir una espiral de asteriscos comenzando en el centro.
10. Elegir tres programas vistos en los capítulos anteriores y con ayuda de un menú calcular tres funciones donde utilice uno, dos y tres parámetros respectivamente.
11. Con ayuda de un menú y 5 funciones, realizar las siguientes tareas:

    a) Llenar e imprimir un vector con números aleatorios entre 25 y 75.

    b) Llenar las columnas impares de una matriz con números aleatorios entre 1 y 50.

    c) Llenar las columnas pares de una matriz con números aleatorios entre 51 y 100.

    d) Imprimir la matriz generada y los elementos de la matriz que estén contenidos en el vector.

12. En el ejercicio 11 complementario en lenguaje C de matrices, utilizar un menú con 4 opciones (una función por cada inciso).
13. Elevar cualquier número a cualquier potencia utilizando la recursividad.
14. Generar la sucesión de Fibonacci utilizando la recursividad.

5.9 Funciones predefinidas de lenguaje C

Los lenguajes C y C++, como la mayoría de los lenguajes de programación, permite el uso de "bibliotecas" con *funciones predefinidas* que se pueden utilizar en cualquier programa. Sólo las funciones más comunes vemos en las tablas 5.1, 5.2, 5.3, 5.4, 5.5, 5.6 y 5.7.

En esta sección sólo presentaremos los ejercicios en lenguaje C, ya que las funciones predefinidas son propias del lenguaje. En la tabla 5.1 comenzaremos con las funciones matemáticas.

Tabla 5.1 Funciones matemáticas (librería *math.h*)

Nombre	*Descripción*	*Tipo de argumentos*	*Tipo de valor de regreso*	*Ejemplo*	*Valor que regresa*
abs(x)*	Valor absoluto de un int	int	int	abs(−7) o abs(7)	7
acos(x)	Arco coseno de x	double	double**	acos(0.7071)*180/3.1416)	45.0004
asin(x)	Arco seno de x	double	double**	asin(0.7071)*180/3.1416)	44.9993
atan(x)	Arco tangente de x	double	double**	atan(1)*180/3.1416)	44.9998
ceil(x)	Redondea hacia el número inmediato superior	double	double	ceil(5.01) ceil(5.9)	6.0
cos(x)	Coseno de x en radianes	double	double	cos(45*3.1416/180))	0.7071
exp(x)	Exponencial de x, e^x .	double	double	exp(1)	2.7183
fabs(x)	Valor absoluto de un double	double	double	fabs(−7.5) fabs(7.5)	7.5
floor(x)	Redondeo hacia el número inmediato inferior	double	double	floor(5.01) floor(5.9)	5.0
fmod(x,y)	Residuo de x/y como número de punto flotante	double	double	fmod(−3.6314, 2.2540)	−1.3774
log(x)	Logaritmo natural de x	double	double	log (10)	2.3026
log10(x)	Logaritmo en base 10 de x	double	double	Log10(100)	2.0
pow(x,y)	Potencia x^y	double	double	pow(2.0,3.0)	8.0
sin(x)	Seno de x en radianes	double	double	sen(45*3.1416/180))	0.7071
sqrt(x)	Raíz cuadrada de x	double	double	sqrt(25.0)	5.0
tan(x)	Tangente de x en radianes	double	double	tan(45*3.1416/180)	1.0

* La función está incluida en la librería *stdlib.h*.

Nota: En todos los resultados de las funciones trigonométricas se manejaron cuatro decimales y en las funciones restantes un decimal.

5.10 Ejercicios resueltos de funciones predefinidas de lenguaje C

Ejercicio 1. Leer el ángulo en grados y mediante un menú de 3 opciones calcular: s) seno, c) coseno y t) tangente.

```
#define PI 3.1416
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
    char op; float ang, resp;
    printf("menú funciones trigonométricas\n");
    printf(" s)seno\n c)coseno\n t)tangente\n");
    printf("dame la opción: "); scanf("%s",&op);
    printf("dame el ángulo en grados: " ); scanf("%f",&ang);
    switch(op)
    {
        case'S':case's':resp=sin(ang*PI/180); printf("seno= %f",resp);
                        break;
        case'C':case'c':resp=cos(ang*PI/180); printf("coseno= %f",resp);
                        break;
        case'T':case't':resp=tan(ang*PI/180); printf("tangente=%f",resp);
                        break;
        default:printf("la función no existe");
    } getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería: menú funciones trigonométricas

s) Seno

c) Coseno

t) Tangente

dame la opción: s

dame el ángulo en grados: 45

seno= 0.707108

En el ejercicio 1 se declara la variable *op* como *char* debido a que las opciones son de tipo carácter (*s, c, t*); se captura la opción (*op*) y el ángulo en grados. Se elige una opción, se realiza la conversión de grados a radianes y posteriormente se calcula la función elegida; por último se imprime el resultado obtenido.

5.10.1 Funciones para manejo de caracteres y cadenas

Se encuentran en las librerías *string.h* (tabla 5.2), *ctype.h* (tabla 5.3) y *stdlib.h* (tabla 5.4). Varias de las macros y funciones declaradas en <ctype> realizan pruebas y conversiones de caracteres.

En la tabla 5.3, las macros regresan un valor *int* y las funciones un valor *char*.

Operaciones con cadenas de caracteres *string.h*

Tabla 5.2 Librería *string.h*

Nombre	***Descripción***
strcat (cad1,cad2)	Concatena (agrega) *cad2* al final de *cad1* (incluyendo el carácter nulo); el carácter inicial de *cad2* sobrescribe el carácter nulo al final de *cad1*. Devuelve el valor de *cad1*.
strcmp(cad1,cad2)	Compara los elementos de dos cadenas *cad1* y *cad2*. Si son iguales, devuelve 0. Si el de *cad1* es mayor que *cad2*, devuelve un valor mayor que cero; en caso contrario, devuelve un valor menor que cero.
strcpy(cad1,cad2)	Copia la cadena *cad2*, incluyendo el nulo, en el arreglo *cad1*. Devuelve el valor de *cad1*.
strlen(cad)	Devuelve el número de caracteres de la cadena *cad*, sin incluir el nulo de terminación.
strncat(cad1,cad2,n)	Concatena *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.
strncmp(cad1,cad2,n)	Es igual que *strcmp* pero con *n* caracteres, compara los *n* primeros caracteres de una cadena.
strncpy(cad1,cad2,n)	Copia los primeros *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.

(*continúa*)

Tabla 5.2 Librería *string.h* (*continuación*)

Nombre	*Descripción*
strchr(cad,c)	Localiza la primera ocurrencia del carácter *c* en la cadena *cad*; si no se encuentra, devuelve un puntero nulo.
strstr(cad1, cad2)	Busca en *cad1* la subcadena *cad2*, si la encuentra devuelve un puntero hacia la primera ocurrencia de *cad2* en *cad1*, si no regresa *null*.
strupr(cad)	Convierte a mayúsculas la cadena *cad*.
strlwr(cad)	Convierte a minúsculas la cadena *cad*.

Ejercicio 2. Leer el Password, si es correcto imprimir "Correcto, puedes continuar" y si no imprimir "Password equivocado".

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
main()
{
   char cad[80];
   puts("Dame tu Password :"); gets(cad);
   if((strcmp(cad,"acceso1234")) == 0)
         puts("Correcto, puedes continuar");
   else puts("Password equivocado");
   getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame tu Password :

acceso1234

Correcto, puedes continuar

En este ejercicio declaramos una cadena llamada *cad* con un máximo de 80 caracteres incluyendo el \0; se utiliza la función puts para imprimir el letrero "Dame tu Password :" y se lee la variable *cad* con la función gets. La función *strcmp* compara la cadena capturada con el letrero "acceso1234", si regresa cero las cadenas son iguales e imprime "Correcto, puedes continuar", si la función *strcmp* regresa un valor diferente de cero las cadenas son diferentes y por lo tanto imprime "Password equivocado".

Pruebas y conversiones de caracteres *ctype.h*

Tabla 5.3 Librería *ctype.h*

Nombre	*Descripción*	*Ejemplo*	*Valor que regresa*
	Macros		
isalpha(c)	Comprueba si un carácter es *alfabético*	isalpha('a') isalpha('7') isalpha('A')	2 0 1
isdigit(c)	Comprueba si un carácter es un *dígito decimal*.	isdigit('a') isdigit('7')	4 0
islower(c)	Comprueba si un carácter es de tipo *minúscula*. Devuelve un valor distinto de cero si *c* es cualquiera de las letras minúsculas [a-z]	islower('a') islower('Z') islower('7')	2 0 0
ispunct(c)	Comprueba si un carácter es *signo de puntuación*.	ispunct('.') ispunct('?')	16 16

(*continúa*)

Tabla 5.3 Librería *ctype.h* (*continuación*)

Nombre	*Descripción*	*Ejemplo*	*Valor que regresa*
isspace(c)	Comprueba si un carácter es un *espacio*	isspace(' ')	8
isupper(c)	Comprueba si un carácter es de tipo *mayúscula*. Devuelve un valor distinto de cero si *c* es cualquiera de las letras mayúsculas [A-Z].	isupper('a') isupper('Z')	0 1
toascii(c)	Convierte *c* a su correspondiente código ASCII; regresa un valor dentro del rango entre 0 y 127.	toascii('a') toascii('A')	97 65
	Funciones		
tolower(c)	Devuelve la correspondiente letra minúscula si existe y si isupper(c) es distinto de cero; en caso contrario, devuelve *c*.	tolower('A') tolower('7')	a 7
toupper (c)	Devuelve la correspondiente letra mayúscula si existe y si islower(c) es distinto de cero; en caso contrario, devuelve *c*.	toupper('a') toupper('7')	A 7

Ejercicio 3. Generar el código de César. El cifrado César mueve cada letra un determinado número de espacios en el alfabeto.

```
#include <ctype.h>
#include <string.h>
#include <conio.h>
#include <stdio.h>
main()
{
    char cadena[80], codigo[80], alfab[27];
    int i,j, espacio;
    for (i=0,j= 65;i<26;i++,j++)
        alfab[i] =j;
    printf("Dame el Texto: "); gets(cadena);
    printf("Dame el número de espacios: ");
    scanf("%d",&espacio);
    for (i=0;i<strlen(cadena);i++)
    {
        cadena[i]=toupper(cadena[i]);
            for (j=0;j<26;j++)
                if (cadena[i]==alfab[j]) break;
        j= j+espacio;
        if (j>25) j=j−26;
        codigo[i]=alfab[j];
    }
    codigo[strlen(cadena)]= '\0';
    puts (codigo); getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

Dame el Texto: ABCD

Dame el número de espacios: 2

CDEF

En el ejercicio 3 se declaran las cadenas *cadena*, *codigo* y *alfab*. Además se declaran las variables i, j y espacio como enteras. En el primer *for* se comienza a generar el alfabeto desde la letra A mayúscula (código ASCII 65) hasta completar las 26 letras. Se lee la cadena y el número de espacios que se recorrerá a la derecha. En el *for* anidado se recorre toda la cadena manejando como límite el tamaño de la cadena con la función *strlen*. Con la función *toupper* convertimos cada carácter a su respectiva mayúscula, ya que en el *for* interno se com-

para el carácter que va de la cadena con el alfabeto generado en el primer *for* en mayúsculas. En el momento que encuentra la letra respectiva se recorre en el código ASCII la cantidad de espacios previamente capturada y si la posición j es mayor a 25 se le resta 26 ya que se rebasó la Z y debe regresar a la letra A. La variable *codigo* va generando el código respectivo carácter por carácter; a dicha variable se le agrega \0 en la posición que corresponde al fin de cadena capturada. Por último se imprime el código respectivo.

Ejercicio 4. Leer una cadena e imprimir el total de: consonantes, vocales, dígitos, caracteres especiales, mayúsculas, minúsculas, espacios en blanco y caracteres de la cadena.

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
main ()
{
   char cadena[26];
   int consonan=0,vocales=0,digitos=0,punt=0,may=0,min=0,espacios=0,i, total;
   printf (“Introduce una oración (máx. 25 caracteres): “);
   gets(cadena); total=strlen(cadena);
   for(i=0;i<total;i++)
   {
      if (isdigit(cadena[i])) digitos++; // Cuenta dígitos
      if (ispunct(cadena[i])) punt++; // Cuenta signos de puntuación
      if (isspace(cadena[i])) espacios++; // Cuenta espacios en blanco
      if (isalpha(cadena[i])) // Letras
      {
         if(isupper(cadena[i])) may++; // Cuenta mayúsculas
         else min++; // Cuenta minúsculas
         cadena[i]=tolower(cadena[i]); //Pasamos letras a minúsculas para contarlas
         switch(cadena[i])
         {
            case (‘a’):case (‘e’):case(‘i’):case(‘o’):case(‘u’): vocales++;// cuenta vocales
            break;
            default:consonan++;
         }
      }
   }
   printf (“En la frase hay:\n”);
   printf (“Consonantes:%d\n“, consonan);
   printf (“Vocales :%d\n“ , vocales);
   printf (“Dígitos :%d\n“ , digitos);
   printf (“Signos de puntuación:%d\n “,punt);
   printf (“Mayúsculas :%d \n“,may);
   printf (“Minúsculas :%d \n“,min);
   printf (“Espacios en blanco:%d\n“,espacios);
   printf (“Total:%d\n“,total);getch();return 0;
}
```

Una posible salida en pantalla después de ejecutar el programa, sería:

Introduce una oración (max. 25 caracteres):

Revolución # 1500

En la frase hay;

Consonantes: 5

Vocales: 4

Dígitos: 4

Signos de puntuación: 1 //El carácter #

Mayúsculas: 1

Minúsculas: 9

Espacios en blanco: 2

Total: 17

En este ejercicio declaramos una cadena, varios contadores, la variable *i* que recorre la cadena y la variable *total* (tamaño de la cadena); en seguida capturamos la cadena. El *for* va recorriendo cada carácter de la cadena y con varios *if* simples vamos revisando si es dígito, signo de puntuación o letra; en el último caso revisa si es letra mayúscula o minúscula. Por último con ayuda de un *switch* define si es vocal o consonante. Cuando sale del ciclo *for* imprime todos los contadores y al final el total de caracteres. Las vocales acentuadas son consideradas como consonantes se podrían validar dichas vocales en el case, ya que ahí es considerada como minúscula.

Ejercicio 5. Comprobar si un texto introducido por teclado es o no un palíndromo.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>
main()
{
    char cad1[80],cad2[80];
    int i,c=0;
    printf("Dame una Cadena: "); gets(cad1); strupr(cad1);
    for(i=0;i<strlen(cad1);i++)
        if(!isspace(cad1[i]))
        {
            cad2[c]=cad1[i]; c++;
        }
    cad2[c]='\0';
    strcpy(cad1,cad2);
    //invertimos cadena
    for(i=0;i<strlen(cad1);i++)
    {
        c--; cad2[i]=cad1[c];
    }
    if (strcmp(cad1,cad2)==0) printf("Es palíndroma");
        else printf("No es palíndroma");
    getch(); return 0;
}
```

Una posible salida en pantalla después de ejecutar el programa. sería:

Dame una Cadena: anita lava la tina

Es palíndroma

En este ejercicio declaramos 2 cadenas (*cad1* y *cad2*) y 2 contadores enteros (*i* y *c*). Capturamos la cadena *cad1*. Con el primer *for* eliminamos los espacios en blanco generando la cadena *cad2*. Con la función *strcpy* la variable *cad1* recibe una copia de *cad2*. Invertimos la cadena *cad1* y el valor lo toma *cad2*. Por último con la función *strcmp* comparamos las dos cadenas. En caso de que sean iguales imprime *"Es palíndroma"* y si son diferentes imprime *"No es palíndroma"*.

Ejercicio 6. Leer un número arábigo e imprimir su correspondiente romano.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<stdio.h>
```

i	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
romano	*M*	*CM*	*D*	*CD*	*C*	*XC*	*L*	*XL*	*X*	*IX*	*V*	*IV*	*I*
arabigo	1000	900	500	400	100	90	50	40	10	9	5	4	1

(*continúa*)

(*continuación*)

```
#include<string.h>
main ()
{
int num,i, arabigo[13]={1000,900,500,400,100,90,50,40,10,9,5,4,1};
char romano[13][3]={"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV",
  "I"}, NumRom[15]={"\0"};
printf("Dame un número arábigo: "); scanf("%d",&num);
   while (num!= 0)
   {
     for (i=0;i<13;i++)
         if (num>=arabigo[i])
            {
               strcat(NumRom,romano[i]);
               num=num−arabigo[i];
               break;
            }
   }
   printf("%s\n",NumRom);getch(); return 0,
}
```

Una posible salida en pantalla después de ejecutar el programa sería:

Dame un número arábigo: 1756

MDCCLVI

En este ejercicio se declaran 2 variables enteras *num* (número arábigo) e *i* (recorre el arreglo *arabigo*). Se inicializa el arreglo arabigo con los valores equivalentes en romano, la cadena *NumRom* con \0 y romano con las diferentes etiquetas de los números romanos. Se captura el número romano (*num*). El *while* da vueltas mientras *num* sea diferente de cero, internamente un *for* recorre las 13 posiciones del arreglo *arabigo*, si *num* es mayor o igual al número arábigo que va *NumRom* concatena el valor correspondiente a la etiqueta en esa posición, por ejemplo si *num*=*970* se coloca en la posición *i*=1 y concatena *CM*. El valor de *num* va disminuyendo, en nuestro ejemplo se le restaría 900 ya que ya se imprimió dicho valor en romano. El ciclo *while* termina cuando el valor de *num* sea 0. Al final imprimimos *NumRom*.

Ejercicio 7. Cambiar las vocales de 3 cadenas por un determinado carácter.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
main ()
{
   char cadena[3][70], c;
   int i, j;
   printf ("Dame un carácter : ");
   scanf("%c",&c);
   for (i = 0; i < 3; i++)
   {
       printf ("Cadena %d: ", i+1);
       fflush(stdin);
       gets (cadena[i]);
   }
```

Una posible salida en pantalla después de ejecutar el programa, sería:

Dame un carácter : $

Cadena 1: La casa

Cadena 2: CUCEI

Cadena 3: Programacion

CADENAS NUEVAS

L$ cs

C$C$$

Prgrm$c$$n

(*continúa*)

(*continuación*)

```
    puts("\nCADENAS NUEVAS");
    for (i = 0; i< 3; i++)
    {
        for (j = 0; cadena[i][j]; j++)
            if (strchr("aeiouAEIOU",cadena[i][j]))cadena[i][j]=c;
        puts (cadena[i]);
    }
    getch(); return 0;
}
```

En este ejercicio se declara una matriz *cadena* (manejaremos 3 cadenas) y carácter *c*. Capturamos las 3 cadenas en el primer *for*. En el segundo *for* se recorre todos los caracteres de las cinco cadenas, con la función *strchar* revisamos si el carácter que va en ese momento corresponde a una vocal minúscula o mayúscula, de ser así lo sustituye por el carácter *c* capturado. Cada vez que termina con una cadena la imprime ya modificada con la función *puts*.

Ejercicio 8. Con la función *strncpy* podemos crear una nueva cadena con *n* caracteres de la cadena anterior, por ejemplo.

Ejemplos



```
#include <stdio.h>
#include <string.h>
#include <conio.h>
main()
{
  char cad_ant[30],
  cad_nva[30]="programación";
  int n;
  printf("Dame una cadena: ");
  gets (cad_ant);
  printf("Dame el total de caracteres que quieres copiar, comenzando del primero: ");
  scanf("%d",&n);
  strncpy(cad_nva, cad_ant, n);
  printf ("La nueva cadena es: %s",cad_nva);
  getch(); return 0;
}
```

Una posible salida en pantalla después de ejecutar el programa sería:

Dame una cadena: CUCEI

Dame el total de caracteres que quieres copiar,

comenzando del primero: 3

La nueva cadena es: CUCgramación

Operaciones con cadenas de caracteres *stdlib.h*

Tabla 5.4 Funciones para la conversión de tipos, librería *stdlib.h*

Nombre	*Descripción*	*Tipo de valor de regreso*	*Ejemplo*	*Valor que regresa*
atof(cad)	Convierte los caracteres de la cadena *cad* a la representación interna de tipo *double* y devuelve ese valor.	double	atof("25.5")	25.500000
atoi(cad)	Convierte los caracteres de la cadena *cad* a la representación interna de tipo *int* y devuelve ese valor.	int	atoi("25.5")	25

Nota: En las funciones atof y atoi, si el primer carácter de la cadena no es un número devuelven 0.

Ejercicio 9. Validar que el valor capturado sea un número entero.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
main()
{
   char cad[]="\0"; int num;
   printf("Dame un número diferente de cero: ");
   gets(cad);
   num= atoi(cad);
   if (num!=0) printf("El número es: %d",num);
       else printf("Lo que capturaste NO es un número");
   getch(); return 0;
}
```

Una posible salida en pantalla después de ejecutar el programa sería:

Dame un número diferente de cero: 1235t5

El número es: 1235

En este ejercicio la función *atoi* toma el número hasta que se encuentra un carácter que no sea numérico.

5.10.2 Funciones de pantalla

Librería *conio.h (Turbo C)/ conio2.h ()*

La tabla 5.5 contiene las funciones de E/S de cadenas, funciones de teclado, posición de pantalla, etc. Entre las más utilizadas tenemos:

Tabla 5.5 Librería *conio.h*

Nombre	*Descripción*
void clrscr();	Borra toda la pantalla. Sólo se aplica cuando la pantalla está en modo de texto.
void gotoxy(int x, int y);	Ubica el cursor en las coordenadas x (columna), y (renglón). Esta operación sólo se utiliza en pantallas de modo de texto.
void highvideo(void);	Aumenta la intensidad de la luminosidad de los caracteres. Afecta a todos los caracteres que se imprimen posteriormente en pantalla dentro de esa ventana.
void normvideo(void);	Anula a highvideo() y/o normvideo(). Es decir, la luminosidad de los caracteres que se escriban a continuación, será normal dentro de esa ventana.
int kbhit();	Revisa si se presionó una tecla. Retorna 0 si no se ha pulsado una tecla.
void textcolor(int color);	Elige un nuevo color para los caracteres que se escribirán a continuación.
void texbackground(int color);	Elige un nuevo color del fondo de la pantalla.
void window(int inix,int iniy,int finx,int finy);	Crea una ventana activa en pantalla.

Ejercicio 10. Introducir un carácter e imprimirlo en el centro de la pantalla; posteriormente imprimirlo en diferentes posiciones de la pantalla con las flechas de movimiento de cursor. Para terminar oprimir la tecla ESC (escape).

```
#include<stdio.h>
#include<conio2.h>
```

(*continúa*)

(*continuación*)

```
main ()
{
    int c=40,r=12;
    char tecla,car;
    printf("Dame un carácter: ");
    car=getche();
    do
    {
        gotoxy(c,r);
        printf("%c",car);
        tecla=getch();
        switch(tecla)
        {
            case 72:if (r>1) r--;
                        else printf("\a");
                    break;
            case 75:if (c>1) c--;
                        else printf("\a");
                    break;
            case 77:if (c<80) c++;
                        else printf("\a");
                    break;
            case 80:if (r<=24) r++;
                        else printf("\a");
                    break;
        }
    }
    while (tecla!=27); return 0;
}
```

Como en este ejercicio utilizamos la librería conio2, los compiladores utilizados son DEV-CPP y Code Blocks. Se declaran e inicializan dos variables enteras *c=40* y *r=12*, la primera corresponde a la columna y la segunda al renglón; recordemos que la pantalla en modo de texto normalmente tiene 80 columnas y 24 renglones utilizables (dependiendo de la configuración puede tener 48 renglones). Se captura el carácter y entra al ciclo *while* donde en la primera vuelta el carácter aparecerá en el centro de la pantalla (40,12). Se oprime una tecla y con ayuda de un switch se determina si el movimiento fue hacia arriba (código 72), a la izquierda (código 75), a la derecha (código 77) o hacia arriba (código 80); en cada caso se delimitará con el tamaño de la pantalla, por ejemplo si oprimimos varias veces hacia la derecha y el contado de columnas *c* es de 80 mandará un alerta ya que no puede seguir; lo mismo ocurrirá en los otros tres movimientos. A partir de la segunda vuelta del ciclo *while* se imprimirá el carácter con el movimiento indicado. El ciclo *while* termina cuando oprimamos la tecla de escape (código 27).

Ejercicio 11. Mostrar las 10 tablas de multiplicar en una pantalla.

En este ejercicio utilizamos dos ciclos anidados *do-while*, el externo *t* controla desde la tabla del 1 a la del 10 y el interno *i* controla del 1 al 10 en cada tabla. Declaramos tres variables de tipo entero *i, t* y *c*; las últimas dos se inicializan con 1 debido a se que comienza con la tabla del 1 y en la columna 1. El ciclo externo se inicializa con 1 (primera tabla), entra al ciclo interno y verifica si (*t*<6), como es así imprime *1*1=1* y la variable *i* se incrementa en 1, por lo que en el *gotoxy* tendremos (1,2) imprimiendo en la columna 1, renglón 2 y así sucesi-

vamente hasta llegar al renglón 10, cuando *i=11* se rompe el ciclo interno. Las primeras 5 tablas se imprimen de la mitad de la pantalla hacia arriba. La variable c se incrementa en 15 (ancho donde se imprimirá cada tabla), cuando ya se haya impreso la tabla del 5, el contador de columnas *c* comienza en 1, ya que las tablas del 6 al 10 se imprimen de la mitad de la pantalla hacia abajo. En seguida el contador *t* se incrementa en 1, *t=2*, se revisa la condición *(2<=10)* y como es verdadera da otra vuelta, verifica si la *t* (la tabla) es menor que 6, como es verdadero repite todo el proceso anterior hasta que *t=6*; en lugar de utilizar el *gotoxy (c,i)* se ejecuta el *gotoxy (c,i+12)* ya que las tablas del 6 al 10 se imprimen a partir del renglón 13 en adelante.

```
#include<stdio.h>
#include<conio2.h>
main()
{
   int i, t=1,c=1;
   do
   {
       i=1;
       do
       {
            if (t<6) gotoxy(c,i);
               else gotoxy(c,i+12);
            printf("%d*%d=%d",t,i,t*i);
            i++;
       }while(i<=10);
       c=c+15;
       if (t==5)c=1;
       t++;
   }while(t<=10); getch(); return 0;
}
```

Ejercicio 12. Simular un reloj en el centro de la pantalla y terminar con la tecla de escape.

```
#include<conio2.h>
#include<stdio.h>
#include<windows.h>
main()
{
   int min,seg,cent;
   char tecla;
   for(min=0;min<60;min++)
       for(seg=0;seg<60;seg++)
           for(cent=0;cent<100;cent++)
           {
               if (kbhit())//preguntamos si se presionó alguna tecla
               {
                   tecla=getch();//guardamos la tecla presionada
                   if (tecla==27)
                   {
                       min=60;seg=60;cent=100;
```

(*continúa*)

(*continuación*)

```
                    }
                }
                gotoxy(32,12);
                printf("%2d:%2d:%2d",min,seg,cent);
                Sleep(100); //retarda 100 milisegundos, aunque depende de la
                  velocidad del procesador
            } return 0;
}
```

En este ejercicio declaramos tres variables enteras *min, seg* y *cent*. Utilizamos tres ciclos anidados, el externo controla los minutos, el intermedio los segundos y el interno las centésimas de segundo. Debemos recordar que en los ciclos anidados, todos los ciclos se inicializan con un valor y tiene que terminar (romperse) el más interno para que le dé permiso de incrementarse al que está afuera de él. En este caso los tres ciclos se inicializan con cero cuando entra al más interno (*cent*), se espera a que se oprima una tecla con ayuda de la función *kbhit*, hasta que se oprima la tecla de escape (código 27); si no se presiona alguna tecla o se elige otra diferente los ciclos continúan trabajando. En el momento que se oprime la tecla de escape los tres contadores toman los valores que provocan que se rompan los tres ciclos (min=60, seg=60 y cent=100). En este ejercicio se utilizó la función *Sleep* (propia de DEV-CPP y Code::Blocks), la cual me permite retardar la impresión cierta cantidad de milisegundos (dependiendo de la velocidad de procesador); dicha función requiere de la librería windows.h. Debemos recordar que en Turbo C en lugar de la función *Sleep* se utiliza *delay*, la cual requiere de la librería *dos.h.*

En la tabla 5.6 vemos las funciones para manejar números aleatorios en *Turbo C.*

Tabla 5.6 Funciones *random* y *rand, librería stdlib.h*

Nombre	*Descripción*
int random (int);	La llamada random (n) devuelve un entero pseudoaleatorio mayor o igual que 0 y menor o igual que n−1. (No está disponible en todas las implementaciones. Si no lo está, debe utilizar *rand*).
int rand();	La llamada *rand()* devuelve un entero pseudoaleatorio mayor o igual que 0 y menor o igual que RAND_MAX. RAND_MAX. Es una constante entera predefinida en *stdlib.h*.

Ejercicio 13. Ejercicio resuelto 5 del capítulo III (hacer-mientras/do-while) en Turbo C donde se utiliza la función *randomize* y *random*.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main()
{
   int nu,nc,c=0;
   randomize();
   nc= random(100) + 1;
   do
   {
       printf("Dame tu número: "); scanf("%d",&nu);
       if (nu > nc) printf("Tu número es mayor al mío.\n");
       if (nu < nc) printf("Tu número es menor al mío.\n");
       c++;
```

(*continúa*)

(*continuación*)

```
    }while (c!=5 && nu!=nc);
        if (nu==nc) printf("FELICIDADES, LO LOGRASTE EN %d INTENTOS!",c);
            else printf("Lo siento el número era %d",nc);
    getch(); return 0;
}
```

Ejercicio 14. Genere 10 números aleatorios con la función *rand* y *random*.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
main()
{
    int aleatorio,i;
    printf("Obteniendo serie de números\n");
    randomize();
    for (i=1; i<=10; i++)
    {
        aleatorio=rand()%10;/*obtiene un numero comprendido entre 0 y 9*/
        printf("%d,%d\n",random(10),aleatorio);
    } getch(); return 0;
}
```

En este ejercicio se declaran 2 variables enteras *aleatorio* e *i.* La primera es el número aleatorio en cuestión y la segunda es el contador de números aleatorios (de 1 a 10). La función *randomize* la utilizamos para crear una semilla que nos garantiza que en cada ejecución los números generados serán diferentes. Cada vez que da vuelta el *for* con la función *rand()%10* se generan 10 números aleatorios comprendidos entre 0 y 9, asignándole el valor a la variable aleatorio para posteriormente imprimir dicho valor.

Función *textcolor ()*

El parámetro color_de_carácter es un número comprendido entre 0 y 15. Cada uno de estos números tiene asociado un color; en el archivo *conio.h* se define una macro para cada color; la relación de colores la apreciamos en la tabla 5.7:

Tabla 5.7 Función *textcolor*

Valor	Color	Macro
0	Negro	BLACK
1	Azul	BLUE
2	Verde	GREEN
3	Turquesa	CYAN
4	Rojo	RED
5	Morado	MAGENTA
6	Marrón	BROWN
7	Blanco	LIGTHGRAY
8	Gris	DARKGRAY
9	Azul intenso	LIGTHBLUE

(*continúa*)

Tabla 5.7 Función *textcolor* (*continuación*)

Valor	Color	Macro
10	Verde intenso	LIGTHGREEN
11	Turquesa intenso	LIGTHCYAN
12	Rojo intenso	LIGTHRED
13	Morado intenso	LIGTHMAGENTA
14	Amarillo	YELLOW
15	Blanco intenso	WHITE
128	Parpadeo	BLINK

Para que un texto se muestre, por ejemplo, en color rojo, antes de su impresión debe anotarse textcolor (4); o bien textcolor (RED). La función textcolor no trabaja en las funciones *printf(), puts(), gets()* y *putchar();* en su lugar, deben utilizarse las funciones *cprintf(), cputs(), cgets()* y *putch()*, respectivamente. La función *getche()* es permitida. Para el parpadeo del carácter debe utilizarse una operación OR entre el color y el valor 128 (BLINK);

> Si queremos presentar un texto en azul parpadeante debemos escribir textcolor (BLUE | BLINK). La instrucción BLINK no se puede utilizar en DEV-CPP ni en Code::Blocks.

Función *textbackground ()*

Establece el color de fondo para todo texto que se escriba en pantalla a continuación: textbackground (color_de_fondo); el color_de_fondo es un valor comprendido entre 0 y 7, correspondiente a la tabla anterior.

Nota: En los compiladores DEV-CPP y Code::Blocks en lugar de la librería *conio.h* tenemos que utilizar *conio2.h* para poder manejar tanto *textcolor()* como *textbackground()*.

Ejercicio 15. Imprimir una espiral en la pantalla con asteriscos a color y en cámara lenta.

Versión en Turbo C

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
void marco (int w, int x, int y, int z)
{ int i; i=x;
   while (i<=80−w)
   {
       textcolor(i); gotoxy(i,1+w);
       cprintf("*"); delay(10); i++;
   }
   i=x;
   while (i<=24−w)
   {
       textcolor(i); gotoxy(80−w,i);
       cprintf("*"); delay(10); i++;
   }
```

Versión en DEV-CPP y Code::Blocks

```
#include <stdio.h>
#include <conio2.h>
#include <windows.h>
void marco (int w,int x,int y,int z)
{ int i; i=x;
    while (i<=80−w)
    {
       textcolor(i); gotoxy(i,1+w);
       cprintf("*"); Sleep(10);
         i++;
    }
    i=x;
    while (i<=24−w)
    {
       textcolor(i); gotoxy(80−w, i);
       cprintf("*"); Sleep(10);
         i++;
    }
```

(*continúa*)

(*continuación*)

Versión en Turbo C	Versión en DEV-CPP y Code::Blocks
i=y;	i=y;
while (i>=1+w)	while (i>=1+w)
{	{
textcolor(i); gotoxy(i,24−w);	textcolor(i); gotoxy(i,24−w);
cprintf(“*”); delay(10); i−−;	cprintf(“*”); Sleep(10); i−−;
}	}
i=z;	i=z;
while (i>=2+w)	while (i>=2+w)
{	{
textcolor(i); gotoxy(1+w,i);	textcolor(i); gotoxy(1+w,i);
cprintf(“*”); delay(10); i−−;	cprintf(“*”); Sleep(10); i−−;
}	}
}	}
main()	main()
{	{
int x,ini=1,fin1=80,fin2=24;clrscr();	int x,ini=1,fin1=80,fin2=24; clrscr();
for (x=0;x<=12;x+=1)	for (x=0;x<=12;x+=1)
marco(x,ini+x,fin1−x,fin2−x);	marco(x, ini+x, fin1−x, fin2−x);
getch();	getch();
}	}

5.11 Creación de librerías en lenguaje C

En lenguaje C puede crear sus propias librerías. En un archivo con extensión *.h* se guardan las funciones que se van a utilizar; dichos archivos generalmente los encontramos en la carpeta *include*.

Ejemplos

Si el archivo se llama *Milibreria.h* y contiene la función:

```
int resta (int a, int b)
{
   return a−b;
}
```

Para utilizar la librería tenemos el siguiente programa:

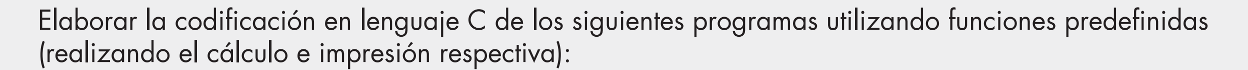
```
#include <conio.h>
#include <stdio.h>
#include "Milibreria.h"
main()
{
   printf("Resta=%d",resta(7,5));
   getch(); return 0;
}
```

Si ejecutamos el programa anterior aparecerá:

Resta=2

Ejercicios complementarios de funciones predefinidas en lenguaje C





Elaborar la codificación en lenguaje C de los siguientes programas utilizando funciones predefinidas (realizando el cálculo e impresión respectiva):

1. Con ayuda de un menú elabore un programa donde utilice:

   a) Cinco funciones matemáticas.

   b) Dos funciones con vectores.

   c) Dos funciones con matrices.

   d) Dos funciones afines a su carrera.

   e) Cinco funciones con el cálculo de diferentes volúmenes (de tal manera que utilice uno, dos y tres parámetros).

   f) Tres funciones de cadena.

   g) Dos funciones donde combine vectores y matrices.

   En los primeros tres incisos utilice funciones sin parámetros y en los últimos incisos funciones con parámetros.

2. Crear una función que contabilice el número de veces que aparece una palabra en una frase.
3. Graficar la función seno o coseno utilizando la función *gotoxy*.
4. Validar que los datos capturados no tengan letras y si tienen letras deben pedirse de nuevo los datos.
5. Leer una cadena de caracteres e imprimir las consonantes en mayúsculas y sus vocales en minúsculas.
6. Leer una cadena e imprimir la cantidad de palabras que tiene una frase.
7. Leer una cadena e imprimir la palabra de tamaño mayor.
8. Leer una cadena y sustituir los espacios en blanco por el carácter &.
9. Leer una cadena y contar el total de las letras a, e, i, o y u.
10. Leer un número entero de hasta 10 dígitos e imprimir por cada digito, el nombre del número correspondiente, separados por una coma. Por ejemplo: 2697 dos, seis, nueve, siete.
11. Leer una secuencia de letras y mostrar la suma de sus códigos ASCII.
12. Leer un número *num* real positivo e intercambiar la parte entera con la parte decimal con el orden de los dígitos invertidos: Si $num = 521.12865$, $ninv = 56821.125$
13. Leer una cadena que representa un número romano menor o igual a 3000 y convertirlo a su equivalente decimal. Validar si es un número romano o no.

    Donde: I=1, V=5, X=10, L=50, C=100, D=500, M=1000

    Las reglas son:

    - Si un número romano compuesto tiene un número a la derecha menor que el de la izquierda entonces se suman ambos.
    - Si un número romano compuesto tiene un número a la derecha mayor que el de la izquierda y éste es un I, X o C, entonces se resta el de la izquierda al de la derecha.
    - Los símbolos I, X, C y M pueden repetirse hasta tres veces. Los símbolos V, L y D sólo pueden aparecer una vez.

14. Leer el nombre y fecha de nacimiento de una persona y obtener su RFC.
15. Crear una librería con los ejercicios 6, 7 y 8; posteriormente utilícela en un programa.



Capítulo 6

Registros o estructuras

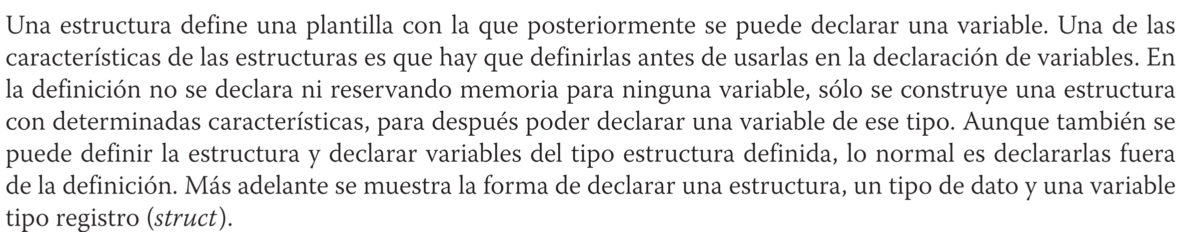
Al término de este capítulo, el alumno será capaz de

- Organizar la información en registros (*struct*), que permitan un manejo integral y con esto facilitar el uso de los datos.
- Manipular los distintos tipos de datos almacenados en una misma estructura (*struct*), diseñando algoritmos y programas en los que tenga acceso a los miembros de un registro.

Contenido

Un registro o estructura es un tipo de dato estructurado y definido por el usuario que permite almacenar datos de diferente tipo en una sola variable; dichos datos pueden ser simples (caracteres, números enteros o de coma flotante, etc.) o compuestos (vectores, estructuras, listas, etc.). A los datos del registro se les denomina campos, elementos o miembros. Una estructura está formada por variables que tienen relación entre sí. También se llaman registros a los grupos de datos en la terminología de las bases de datos.

6.1 Definición de una estructura



Una estructura define una plantilla con la que posteriormente se puede declarar una variable. Una de las características de las estructuras es que hay que definirlas antes de usarlas en la declaración de variables. En la definición no se declara ni reservando memoria para ninguna variable, sólo se construye una estructura con determinadas características, para después poder declarar una variable de ese tipo. Aunque también se puede definir la estructura y declarar variables del tipo estructura definida, lo normal es declararlas fuera de la definición. Más adelante se muestra la forma de declarar una estructura, un tipo de dato y una variable tipo registro (*struct*).

Para crear una estructura primero comenzamos por definir el tipo de estructura, para ello se procede de manera parecida a la definición de una variable, con algunas modificaciones; en lenguaje C para definir una estructura, se utiliza la palabra reservada *struct*, normalmente seguida por un nombre y llave izquierda; después se define el tipo y nombre para uno o más campos. Se permite cualquier definición de tipo habitual, incluso punteros y estructuras. Cada definición de campo acabará con un punto y coma. Por último se coloca una llave derecha. Si hay campos del mismo tipo, se pueden declarar en la misma línea, separándolos por comas.

Hay varias formas de definir un registro, pero la más general es la siguiente:

Pseudocódigo	Lenguaje C
registro ident_registro	`struct ident_registro`
inicio	`{`
tipodato_1 campo_1, campo_2	`tipodato_1 campo_1, campo_2;`
tipodato_2 campo_3, campo_4	`tipodato_2 campo_3, campo_4;`
tipodato_3 campo_5	`tipodato_3 campo_5;`
fin	`};`

Hemos definido el *tipo* de dato *ident_registro* (nombre del registro), pero aún no hay ninguna variable declarada con este nuevo tipo. Observe en lenguaje C la necesidad de incluir un carácter (;) después de cerrar las llaves. Para definir una variable de tipo registro se declara de la misma forma que una variable:

Pseudocódigo	Lenguaje C
ident_registro ident_variable	`ident_registro ident_variable;`

Ejemplos



Pseudocódigo	Lenguaje C
registro info	`struct info`
inicio	`{`
caracter cod_est[9], nom_est[40]	`char cod_est[9], nom_est[40];`
entero sem_est, cod_carr	`int sem_est, cod_carr ;`
real prom_est	`float prom_est;`
fin	`}`
principal()	`main()`
inicio	`{`
info estudiante	`info estudiante;`

(*continúa*)

(*continuación*)

donde:		
	info	nombre del registro
	estudiante	nombre de la variable

También podemos definir variables estructuras sin tipo específico, para ello basta omitir el identificador del tipo de estructura en la definición de la estructura, dando sólo el identificador de la variable estructura. De este modo la nueva variable va asociada al tipo creado. En ese caso, quizá no sea necesario dar un nombre a la estructura, con la siguiente sintaxis:

Pseudocódigo	Lenguaje C
registro	`struct`
inicio	`{`
tipodato_1 campo_1, campo_2	`tipodato_1 campo_1, campo_2;`
tipodato_2 campo_3, campo_4	`tipodato_2 campo_3, campo_4;`
tipodato_3 campo_5	`tipodato_3 campo_5;`
fin	
nomb_var	`}nomb_var;`

Queda así definida la variable *nomb_var*, de tipo de dato *struct*.

En este caso, no se pueden crear nuevas variables de ese tipo, por lo que no se recomienda usar este tipo de declaraciones.

Otra manera de generar la estructura y a la vez declarar las primeras variables de ese tipo, será la siguiente sintaxis:

Pseudocódigo	Lenguaje C
registro ident_registro	`struct ident_registro`
inicio	`{`
tipodato_1 campo_1, campo_2	`tipodato_1 campo_1, campo_2;`
tipodato_2 campo_3, campo_4	`tipodato_2 campo_3,  campo_4;`
tipodato_3 campo_5	`tipodato_3 campo_5;`
fin	`} nomb_var_1,…., nomb_var_n;`
nomb_var_1,…., nomb_var_n	

Se define el registro y declaran las variables *var_1, ..., var_n*, que podrán ser locales o globales.

Usualmente una estructura se define para que pueda manejarse en varias funciones (de modo global). Sin embargo, se puede definir de manera local por lo que la estructura sólo podrá utilizarse en dicha función, por ejemplo:

Ejemplos



Pseudocódigo	Lenguaje C
registro info	`struct info`
inicio	`{`
caracter cod_est[9], nom_est[40]	`char cod_est[9], nom_est[40];`
entero sem_est, cod_carr	`int sem_est, cod_carr ;`
real prom_est	`float prom_est;`
fin estudiante	`} estudiante;`
principal()	`main()`
inicio	`{`
Donde estudiante es una variable global de tipo *info*.	

6.2 Tipos de datos definidos por el usuario *typedef*

En lenguaje C podemos dar un nuevo nombre a tipos de datos que ya existen, a fin de que éstos sean más afines con aquello que representan y haciendo que el código fuente sea más claro. Una vez que se ha creado un tipo de dato y un nombre para hacer referencia a él, podemos usar ese identificador en la declaración de variables, como cualquier tipo de dato estándar en C. Para definir un tipo de dato se utiliza la palabra reservada *define_tipo (typedef)* con el siguiente formato:

Pseudocódigo	Lenguaje C
define_tipo ident_tipo ident_tipo_nuevo	`typedef ident_tipo ident_tipo_nuevo;`

Después, los identificadores se pueden usar para declarar variables y funciones.

Ejemplos



Pseudocódigo	Lenguaje C
define_tipo float kg	`typedef float kg;`
define_tipo float mts	`typedef float mts;`
kg peso	`kg peso;`
mts longitud	`mts longitud;`

Pseudocódigo	Lenguaje C
	#include <stdio.h>
	#include <conio.h>
principal( )	main( )
inicio	{
define_tipo entero enteros	typedef int enteros;
enteros i,j	enteros i,j;
define_tipo enteros lista[5]	typedef enteros lista[5];
define_tipo enteros matriz[2][2]	typedef enteros matriz[2][2];
lista vec←{1,2,3,4,5}	lista vec={1,2,3,4,5};
matriz a←{1,3,5,7}, b←{2,4,6,8}, c	matriz a={1,3,5,7}, b={2,4,6,8}, c;
desde (i←0,i<5,i←i+1)	for (i=0;i<5;i++)
imprimir vec[i]	printf("%d ",vec[i]);
	printf("\n");
for (i←0,i<2,i←i+1)	for (i=0;i<2;i++)
inicio	{
desde (j←0,j<2,j←j+1)	for (j=0;j<2;j++)
inicio	{
c[i][j] ←a[i][j]+b[i][j]	c[i][j]=a[i][j]+b[i][j];
imprimir c[i][j]	printf("%d ",c[i][j]);
fin	}
	printf("\n");
fin	}
	getch(); return 0;
fin	}

Como podemos ver en los ejemplos anteriores *typedef* nos sirve para crear "sinónimos" de tipos de datos ya definidos.

Una manera fácil de definir estructuras en lenguaje C es mediante la combinación de la palabra *struct* y de la palabra *typedef*. La sintaxis puede ser de dos formas, la primera es la siguiente:

Pseudocódigo	Lenguaje C
define_tipo ident_registro ident_tipo_nuevo	`typedef ident_registro ident_tipo_nuevo;`

Por ejemplo, se define una estructura para almacenar los datos básicos de un estudiante que son su código, nombre, código de la carrera, semestre, promedio.

Ejemplos



Pseudocódigo	Lenguaje C
registro info	`struct info`
inicio	`{`
caracter cod_est[9], nom_est[40]	`char cod_est[9], nom_est[40];`
entero sem_est, cod_carr	`int sem_est, cod_carr ;`
real prom_est	`float prom_est;`
fin	`}`
define_tipo info tipo_estud	`typedef info tipo_estud;`
principal()	`main()`
inicio	`{`
tipo_estud estudiante	`tipo_estud estudiante;`

donde:

info	nombre del registro
tipo_estud	nombre de un tipo de dato definido por el usuario
estudiante	nombre de la variable

La segunda manera de combinar una *struct* y un *typedef*, es la siguiente:

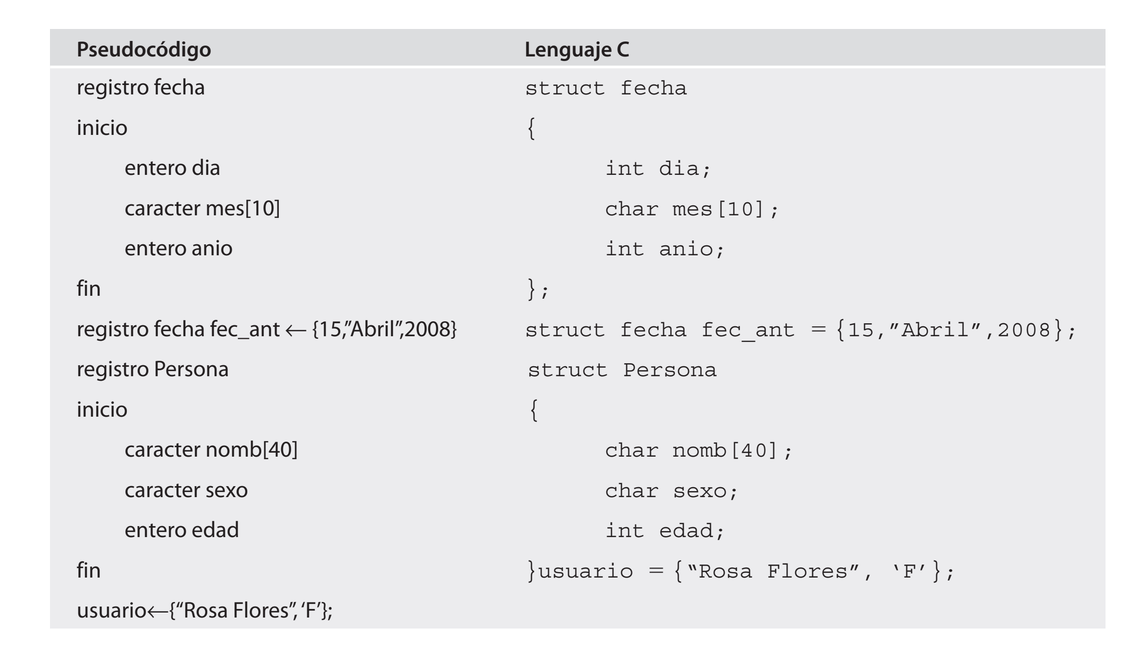
Pseudocódigo	Lenguaje C
define_tipo registro	`typedef struct`
inicio	`{`
tipodato_1 campo_1, campo_2	`tipodato_1 campo_1, campo_2;`
tipodato_2 campo_3, campo_4	`tipodato_2 campo_3, campo_4;`
tipodato_3 campo_5	`tipodato_3 campo_5;`
fin	`}nomb_var;`
nomb_var	

6.3 Asignación de valores a los campos de una estructura

Una vez declarada una estructura, pueden asignárseles valores iniciales a sus campos. Para ello se dan los valores iniciales escribiendo entre llaves los valores de sus campos en el mismo orden en que se declararon éstos, al igual que hacemos con los arreglos. Pero ahora cada dato puede tener un tipo diferente.

Ejemplos





Pseudocódigo	Lenguaje C
registro fecha	`struct fecha`
inicio	`{`
entero dia	`int dia;`
caracter mes[10]	`char mes[10];`
entero anio	`int anio;`
fin	`};`
registro fecha fec_ant ← {15,"Abril",2008}	`struct fecha fec_ant ={15,"Abril",2008};`
registro Persona	`struct Persona`
inicio	`{`
caracter nomb[40]	`char nomb[40];`
caracter sexo	`char sexo;`
entero edad	`int edad;`
fin	`}usuario ={"Rosa Flores", 'F'};`
usuario←{"Rosa Flores", 'F'};	

Como no se proporciona valor para la edad, ésta se inicializa en cero.

6.4 Acceso a los campos de un registro o estructura (*struct*)

Para acceder a cada campo de una estructura se usa el operador punto '.', precedido por el nombre de la *estructura* y seguido del nombre del *campo*. Dicho operador, que sirve para representar la dependencia de cada uno de los miembros con su estructura, es necesario, ya que al declararse varias estructuras del mismo tipo, debemos distinguir los correspondientes campos de unas y otras. Cada campo de una estructura designado mediante este operador se comporta como si se tratase de una variable del mismo tipo que el campo. Podemos realizar todas las operaciones habituales de las variables.

Se accede a los elementos de estructura usando el siguiente formato general:

Pseudocódigo	Lenguaje C
ident_variable.nombre_campo	`ident_variable.nombre_campo`

El tratamiento de cada uno de estos campos depende de su tipo (si es un entero lo asignaremos y trataremos como entero, si es un vector como un vector, si es un *string* como un *string*,...)

Hacemos la asignación mediante la siguiente expresión:

Pseudocódigo	Lenguaje C
iden_variable.campo←valor	`iden_variable.campo=valor;`

Para imprimir el campo lo haremos de la siguiente forma:

Pseudocódigo
imprimir ident_variable.campo o imprimircad ident_variable.campo

Sólo hacemos referencia al pseudocódigo, ya que como veremos en el siguiente ejemplo depende del formato del tipo de dato en el caso del *printf*.

Ejemplos



Pseudocódigo	Lenguaje C
	`#include <stdio.h>`
	`#include <conio.h>`
	`#include <string.h>`
registro Persona	`struct Persona`
inicio	`{`
caracter nomb[40]	`char nomb[40];`
entero edad	`long edad;`
caracter sexo	`char sexo;`
fin	`};`
registro Persona usuario	`struct Persona usuario;`
principal ()	`main ()`
inicio	`{`
registro fecha	`struct fecha`
inicio	`{`
entero dia	`int dia;`
caracter mes[10]	`char mes[10];`
entero anio	`int anio;`
fin	`}fec_act = {29,"Abril",2009};`
fec_act ← {29,"Abril",2009}	
usuario.nomb←"Rosa Flores"	`strcpy (usuario.nomb, "Rosa Flores");`
usuario.edad ← 38	`usuario.edad = 38;`
usuario.sexo ← 'F'	`usuario.sexo = 'F';`
imprimir usuario.nomb	`printf ("%s ",usuario.nomb);`
imprimir usuario.sexo	`printf ("%c ",usuario.sexo);`
imprimir usuario.edad	`printf ("%d\n ",usuario.edad);`
imprimir fec_act.dia	`printf ("%d ",fec_act.dia);`
imprimir fec_act.mes	`printf ("%s ",fec_act.mes);`
imprimir fec_act.anio	`printf ("%d ",fec_act.anio);`
	`getch();`
	`return 0;`
fin	`}`

Tenemos dos registros, el primero llamado Persona es global y el segundo fecha es local a principal (main). Se realizan tres asignaciones una de tipo cadena, otra numérica y la última de un carácter. Si dos campos son del mismo tipo, es posible asignar una a la otra. Si dos variables estructura son del mismo tipo, es posible asignar una a la otra. No se pueden asignar una estructura a otra si son de tipos diferentes, ni siquiera si comparten algunos de sus elementos.

Para almacenar en un campo de un registro se le puede solicitar al usuario y utilizar la instrucción *leer* o *leercad*:

Pseudocódigo
leer ident_variable.campo o leercad ident_variable.campo

Al igual que en la impresión de campos sólo hacemos referencia al pseudocódigo, debido a las diferentes posibilidades de entrada de datos en el lenguaje C.



6.5 Arreglos de estructuras

No sólo las cadenas son los únicos tipos de arreglos que pueden formar parte de una estructura. En general, cualquier arreglo puede ser un campo de una estructura.

Ejemplos



Pseudocódigo	Lenguaje C
registro elementos	struct elementos
inicio	{
entero vector[30]	int vector[30];
real media	float media;
fin	};

Las estructuras, al igual que las demás variables, pueden agruparse en arreglos. Para declarar un arreglo de estructura, primero se debe definir una estructura y después declarar una variable de arreglo de ese tipo. A fin de acceder a una estructura concreta, se pone un índice al nombre de la estructura. Como todas las variables de arreglo, los arreglos de estructuras tienen el cero como primer índice.

Ejemplos



Pseudocódigo	Lenguaje C
	#include<stdlib.h>
	#include<stdio.h>
	#include<conio.h>
registro datos	struct datos
inicio	{
caracter nombre[40], calle[30]	char nombre[40], calle[30];
entero codigo;	int codigo;
fin	};
define_tipo datos datos_dir	typedef datos datos_dir;
principal()	main()
inicio	{
datos_dir info_dir[3]	datos_dir info_dir[3];
entero i	int i;
imprimir "REGISTROS"	printf ("REGISTROS\n");
desde (i←0,i<3, i←i+1)	for(i=0;i<3;i++)
inicio	{
imprimir "Escribe el nombre: "	printf ("Escribe el nombre: ");
	fflush(stdin);
leercad info_dir[i].nombre	gets(info_dir[i].nombre);
imprimir "Escribe la calle: "	printf ("Escribe la calle: ");
	fflush(stdin); gets(info_dir[i].calle);
leercad info_dir[i].calle);	printf ("Escribe el código: ");
imprimir "Escribe el código: "	scanf("%d",&info_dir[i].codigo);
leer info_dir[i].codigo	printf ("\n");
fin	}

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
desde (i←0,i<3, i←i+1)	`for(i=0;i<3;i++)`
inicio	`{`
imprimircad info_dir[i].nombre	`puts(info_dir[i].nombre);`
imprimircad info_dir[i].calle	`puts(info_dir[i].calle);`
Imprimir info_dir[i].codigo	`printf("%d",info_dir[i].codigo);`
	`printf("\n\n");`
fin	`} getch(); return 0;`
fin	`}`

El ejercicio anterior utiliza un arreglo de tipo registro o estructura, para esto se crea una estructura denominada *datos* que tiene tres campos, posteriormente se crea un tipo de dato denominado *datos_dir*. En el programa principal se declara un arreglo *info_dir[3]* que almacenará en cada fila los campos: *nombre, calle* y *codigo*, todos éstos se solicitan al usuario. En el ciclo *desde (for)* se almacenan los datos de tres personas desde posición 0 hasta la posición 2. Dentro del ciclo *desde* se solicitan los campos con la instrucción *leercad info_dir[i].nombre* para el nombre, *leercad info_dir[i].nombre* para la calle, y *leer info_dir[i].codigo* para el código. Al final se imprimen todos los datos de los tres usuarios (jugadores).

6.6 Estructuras anidadas

También es posible que los miembros de una estructura sean a su vez estructuras previamente definidas, dando lugar a estructuras anidadas. Por ejemplo:

Ejemplos



Pseudocódigo	Lenguaje C
	`#include <string.h>`
	`#include <conio.h>`
	`#include <stdio.h>`
registro nombre	`struct nombre`
inicio	`{`
caracter nom[25],ap[15],am[15]	`char nom[25],ap[15],am[15];`
fin	`};`
registro persona	`struct persona`
inicio	`{`
registro nombre nom_com	`struct nombre nom_com;`
entero edad	`int edad;`
caracter sexo /* M = masculino, F = femenino */	`char sexo; /* M = masculino, F = femenino */`
fin	`};`
principal ()	`main()`
inicio	`{`
registro persona empleado[50]	`struct persona empleado[50];`
empleado[3].nom_com.nom = "Ana Rosa"	`strcpy(empleado[3].nom_com.nom, "Ana Rosa");`
empleado[3].nom_com.ap = "Flores"	`strcpy(empleado[3].nom_com.ap, "Flores ");`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
empleado[3].nom_com.am,"González"	`strcpy(empleado[3].nom_com.am, "González");`
empleado[3].edad = 34	`empleado[3].edad = 34;`
empleado[3].sexo = 'F'	`empleado[3].sexo = 'F';`
imprimir empleado[3].nom_com.nom	`printf("%s",empleado[3].nom_com.nom);`
imprimir empleado[3].nom_com.ap	`printf("%s",empleado[3].nom_com.ap);`
imprimir empleado[3].nom_com.am	`printf("%s\n",empleado[3].nom_com.am);`
imprimir empleado[3].edad	`printf("%d\n",empleado[3].edad);`
imprimir empleado[3].sexo	`printf("%c\n",empleado[3].sexo);`
	`getch();`
	`return 0;`
fin	`}`

La primera estructura contiene tres elementos tipo cadena, pero no tiene ninguna variable definida, sólo una estructura llamada *nombre*. Para poder utilizar este nuevo tipo de dato debe estar siempre asociado con la palabra clave *struct*. La siguiente definición de estructura contiene tres campos de los cuales el segundo es la estructura previamente definida la cual llamamos *nombre*. La variable de tipo *nombre* se llama *nom_com*, así la nueva estructura contiene tres variables: *nom_com, edad* y *sexo*. Dentro de la función *main ( )* definimos un arreglo llamado *empleado* de tamaño 50 cada posición con la estructura definida por el tipo *persona*. Después les asignamos valores a todos los campos de nuestro registro en la posición 3. Cuando hacemos la asignación a un campo de la estructura anidada necesitamos utilizar dos puntos (un punto para separar cada estructura), por ejemplo:

Ejemplos



empleado[3].nom_com.nom="Ana Rosa" (el primero para la estructura persona y el segundo punto para la estructura nombre). En el caso de la asignación de cadenas es importante recordar que en lenguaje C las tenemos que concatenar con la función *strcpy*. Por último imprimimos los valores previamente asignados.

Una posible salida en pantalla, después de ejecutar el programa sería:

Ana Rosa Flores González

34

F



6.7 Ejercicios resueltos de registros o estructuras (*struct*)

Ejercicio 1. Almacene en un registro o estructura los datos de un jugador de basquetbol, como son: nombre, edad, teléfono, sexo y altura. Que el programa imprima el sexo y la altura del jugador.

Pseudocódigo	Lenguaje C
	`#include<stdlib.h>`
	`#include<stdio.h>`
	`#include<conio2.h>`
registro dato	`struct dato`
inicio	`{`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C
caracter nom[30], tel [12], s	`char nom[30], tel [12], s;`
entero edad	`int edad;`
real alt	`float alt;`
fin	`};`
define_tipo dato dato_jug	`typedef dato dato_jug;`
principal()	`main()`
inicio	`{`
dato_jug jug1	`dato_jug jug1;`
imprimir "Introduce el nombre "	`printf ("Introduce el nombre ");`
	`fflush(stdin);`
leercad jug1.nom	`gets(jug1.nom);`
imprimir "Introduce la edad "	`printf ("Introduce la edad ");`
leer jug1.edad	`scanf("%d",&jug1.edad);`
imprimir "Introduce el teléfono "	`printf ("Introduce el teléfono ");`
	`fflush(stdin);`
leercad jug1.tel	`gets(jug1.tel);`
imprimir "Introduce el sexo "	`printf ("Introduce el sexo ");`
leer jug1.s	`scanf("%c",&jug1.s);`
imprimir "Introduce la altura "	`printf ("Introduce la altura ");`
leer jug1.alt	`scanf("%f",&jug1.alt);`
limpiar_pantalla	`clrscr();`
imprimir jug1.nom	`printf ("%s",jug1.nom);`
imprimir "Tu sexo es",jug1.s	`printf ("Tu sexo es %c",jug1.s);`
imprimir " y tu altura es",jug1.alt	`printf (" y tu altura es %f",jug1.alt);`
	`getch();`
	`return 0;`
fin	`}`

Este ejercicio utiliza un registro o estructura llamado dato, creándose una estructura que tiene cinco campos, posteriormente se crea un tipo de dato denominado *dato_jug*. En el programa principal se declara una variable llamada *jug1*, que almacenará campos: *nom, tel, s, edad* y *alt*, todos éstos se le solicitan al usuario. Al final imprime los campos antes leídos.

Una posible salida en pantalla, después de ejecutar el programa sería:

introduce el nombre Alberto Salas Arredondo

introduce la edad 27

introduce el teléfono 3817-9575

introduce el sexo H

introduce la altura 1.75

Alberto Salas Arredondo tu sexo es H y tu altura es 1.750000

Ejercicio 2. Almacene con base en el ejercicio 1, los datos de un número de jugadores de basquetbol, e imprimir los datos de un jugador específico. Se recomienda utilizar un arreglo de registros.

Pseudocódigo	Lenguaje C (DEV-CPP y Code::Blocks)
	#include<stdlib.h>
	#include<stdio.h>
	#include<conio.h>
registro dato	struct dato
inicio	{
caracter nom[30], tel [12], s	char nom[30], tel [12], s;
entero edad	int edad;
real alt	float alt;
fin	};
define_tipo dato dato_jug	typedef dato dato_jug;
principal()	main()
inicio	{
dato_jug jug[20]	dato_jug jug[20];
entero tot_jug, i	int tot_jug, i;
imprimir”Total de jugadores max 20 ”	printf (”Total de jugadores max 20 ”);
leer tot_jug	scanf(“%d”,&tot_jug);
desde (i←0, i< tot_jug, i←i+1)	for (i=0; i< tot_jug; i++)
inicio	{
imprimir “Introduce el nombre ”	printf (“\nIntroduce el nombre ”);
	fflush(stdin);
leercad jug[i].nom	gets(jug[i].nom);
imprimir “Introduce la edad ”	printf (“Introduce la edad ”);
leer jug[i].edad	scanf(“%d”,&jug[i].edad);
imprimir “Introduce el teléfono ”	printf (“Introduce el teléfono ”);
	fflush(stdin);
leercad jug[i].tel	gets(jug[i].tel);
imprimir “Introduce el sexo ”	printf (“Introduce el sexo ”);
leer jug[i].s	scanf(“%c”,&jug[i].s);
imprimir “Introduce la altura ”	printf (“Introduce la altura ”);
leer jug[i].alt	scanf(“%f”,&jug[i].alt);
fin	}
imprimir “¿De qué jugador deseas sus datos?”	printf(“¿De qué jugador deseas sus datos? ”);
leer i	scanf(“%d”,&i);
imprimir jug[i−1].nom	printf (“%s\n”,jug[i−1].nom);
imprimir jug[i−1].edad	printf (“%d\n”,jug[i−1].edad);
imprimir jug[i−1].tel	printf (“%s\n”,jug[i−1].tel);
imprimir jug[i−1].s	printf (“%c\n”,jug[i−1].s);
imprimir jug[i−1].alt	printf (“%f\n”,jug[i−1].alt);
	getch(); return 0;
fin	}

Este ejercicio utiliza un arreglo de tipo registro o estructura, para esto se crea una estructura denominada *dato* que tiene cinco campos, posteriormente se crea un tipo de dato denominado *dato_jug*. En el programa principal se declara un arreglo llamado *jug[20]* que almacenará en cada fila los campos: *nom, tel, s, edad* y *alt,* todos éstos se solicitan al usuario. Al inicio del programa se pide el total de jugadores y se almacena en

la variable *tot_jug*, el ciclo *desde (for)* llena el arreglo *jug* desde la posición 0 hasta la posición *tot_jug −1*. Dentro del ciclo *desde* se solicitan los campos con la instrucción *leercad jug[i].nom* para el nombre, *leer jug[i].edad* para la edad, y así de manera sucesiva. Después se pregunta ¿De qué jugador deseas sus datos? Al final se imprimen todos los datos del jugador una posición antes, ya que como sabemos los arreglos comienzan en la posición cero.

En compiladores como **Turbo C** se tienen problemas cuando se le asigna un campo tipo *float*, por lo que es recomendable que en lugar de utilizar la función de entrada formateada *scanf* la función *cin* (requiere la librería *iostream.h*). Por lo tanto el programa respectivo quedaría:

Lenguaje C (TURBO C)

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
struct dato
{
    char nom[30], tel [12], s;
    int edad;
    float alt;
};
typedef dato dato_jug;
main()
{
    dato_jug jug[20]; int tot_jug, i;
    printf (“Total de jugadores max 20 “); scanf(“%d”,&tot_jug);
    for (i=0; i< tot_jug; i++)
    {
        printf (“\nintroduce el nombre “); fflush(stdin); gets(jug[i].nom);
        printf (“Introduce la edad “); scanf(“%d”,&jug[i].edad);
        printf (“Introduce el teléfono “); fflush(stdin); gets(jug[i].tel);
        printf (“Introduce el sexo “); scanf(“%c”,&jug[i].s);
        printf (“Introduce la altura “); cin>>jug[i].alt;
    }
    printf(“¿De qué jugador deseas sus datos? “); scanf(“%d”,&i);
    printf (“%s\n”,jug[i−1].nom); printf (“%d\n”,jug[i−1].edad);
    printf (“%s\n”,jug[i−1].tel); printf (“%c\n”,jug[i−1].s);
    printf (“%f\n”,jug[i−1].alt); getch(); return 0;
}
```

Ejercicio 3. Almacene en un arreglo los datos de un inventario de una tlapalería (código, descripción, precio de compra, cantidad), calcular el precio de venta que es el 35% más del precio de compra. Máximo 500 artículos. Imprimir la descripción y el precio de venta de todo el inventario.

Pseudocódigo	Lenguaje C(DEV-CPP y Code::Blocks)
	`#include<stdio.h>`
	`#include<conio.h>`
registro articulo	`struct articulo`
inicio	`{`
entero cod, cant	`int cod, cant;`
caracter desc[20]	`char desc[20];`

(*continúa*)

(*continuación*)

Pseudocódigo	Lenguaje C(DEV-CPP y Code::Blocks)
real prec_com, prec_ven	`float prec_com, prec_ven;`
fin	`};`
define dato articulo articulos	`typedef articulo articulos;`
principal()	`main()`
inicio	`{`
articulos tlap[500]; entero a, i	`articulos tlap[500]; int a, i;`
imprimir "Total de artículos"; leer a	`printf ("Total de artículos"); scanf("%d",&a);`
desde (i←0, i<a, i←i+1)	`for (i=0; i<a; i++)`
inicio	`{`
imprimir"Dame los datos"	`printf("\nDame los datos\n");`
imprimir "Código "; leer tlap[i].cod	`printf("Código "); scanf("%d",&tlap[i].cod);`
imprimir "Descripción "; leer tlap[i].desc	`printf("Descripción "); fflush(stdin);`
	`gets (tlap[i].desc);`
imprimir "Precio de compra "	`printf ("Precio de compra");`
leer tlap[i].prec_com	`scanf("%f",&tlap[i].prec_com);`
imprimir "Cantidad en el inventario "	`printf ("Cantidad en el inventario ");`
leer tlap[i].cant	`scanf("%d",&tlap[i].cant);`
tlap[i].prec_ven← tlap[i].prec_com*1.35	`tlap[i].prec_ven=tlap[i].prec_com*1.35;`
fin	`}`
desde (i←0, i<a, i←i+1)	`for (i=0; i<a;i++)`
inicio	`{`
imprimir tlap[i].desc	`printf ("%s\n",tlap[i].desc);`
imprimir tlap[i].prec_ven	`printf ("%f\n",tlap[i].prec_ven);`
fin	`} getch(); return 0;`
fin	`}`

Este ejercicio utiliza un arreglo de tipo registro o estructura, para esto se crea una estructura denominada *artículo* que tiene 5 campos, después se crea un tipo de dato denominado *articulos*. En el programa principal se declara un arreglo *tlap[500]* que almacenará en cada fila los campos: *cod, desc, prec_com, prec_ven, cant,* todos éstos son solicitados al usuario, excepto el *prec_ven*, ya que éste se calcula de manera directa utilizando el campo *prec_com* incrementándole el 35%, con la siguiente instrucción *tlap[i].prec_ven= tlap[i].prec_com*1.35*. Al inicio del programa se solicita el total de artículos y se almacena en la variable *a*, el ciclo *desde (for)* llena el arreglo *tlap* desde la posición 0 hasta la posición a−1. Dentro del ciclo *desde* se solicitan los campos con la instrucción *leer tlap[i].cod* para el código, *leer tlap[i].desc* para la descripción, y así sucesivamente, excepto el campo *tlap[i].prec_ven* que se calcula de manera directa. Es importante recalcar que la posición *i* del arreglo *tlap*, nos indica el índice de tal forma que permite almacenar en cada posición del mismo.

Ejemplos



Si a= 4 se realizará el ciclo desde con i=0, hasta i=4 y se almacenan los siguientes campos:

		tlap			
	cod	***desc***	***prec_com***	***Cant***	***Prec_ven***
0	11	Martillo	100	5	135.0
1	12	Serrucho	130	4	175.5
2	13	Foco (60 kw)	10	25	13.0
3	14	Extensión eléctrica (5 m)	50	12	67.5

Ejemplos



El programa en Turbo C:

```
#include< iostream.h>
#include<stdio.h>
#include<conio.h>
struct articulo
{
   int cod, cant;
   char desc[20];
   float prec_com, prec_ven;
};
typedef articulo articulos;
main()
{
   articulos tlap[500]; int a, i;
   printf ("total de artículos "); scanf("%d",&a);
   for (i=0; i<a; i++)
   {
      printf ("\nDame los datos\n");
      printf ("Código ");
      scanf("%d",&tlap[i].cod);
      printf ("Descripción ");
      fflush(stdin);
      gets(tlap[i].desc);
      printf ("Precio de compra ");
      cin>>tlap[i].prec_com;
      printf ("Cantidad en el inventario ");
      scanf("%d",&tlap[i].cant);
      tlap[i].prec_ven = tlap[i].prec_com*1.35;
   }
   for (i=0; i<a;i++)
   {
      printf ("%s\n",tlap[i].desc);
      printf ("%f\n",tlap[i].prec_ven);
   }
   getch(); return 0;
}
```

Ejercicio 4. Utilizando el ejercicio anterior, haga dos funciones con paso de parámetros por valor que obtenga la utilidad de la empresa si se vendieran todos los artículos que obtenga la cantidad de artículos del inventario.

Pseudocódigo

```
registro articulo
inicio
   entero cod, cant
   caracter desc[20]
   real prec_com, prec_ven
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
fin
tlap[500]
real utilidad (entero a)
inicio
    entero i
    real util ← 0
    desde (i ← 0, i < a, i ← i+1)
        util ← util+ (tlap[i].prec_ven − tlap[i].prec_com)* cant
    regresa util
fin
entero cantidad_total (entero a)
inicio
    entero i, t_art ← 0
    desde (i ← 0, i < a , i←i+1)
        t_art ← t_art + tlap[i].cant
    regresa t_art
fin
principal()
inicio
    entero a, i, t_art
    real util
    imprimir "Total de artículos"
    leer a
    desde (i ← 0, i<a, i ← i +1)
    inicio
        imprimir"Dame los datos"
        imprimir"Código"
        leer tlap[i].cod
        imprimir "Descripción"
        leer tlap[i].desc
        imprimir "Precio de compra"
        leer tlap[i].prec_com
        imprimir "Cantidad en el inventario"
        leer tlap[i].cant
        tlap[i].prec_ven ← tlap[i].prec_com*1.35
    fin
    imprimir "1) Calcula la utilidad de la empresa"
    imprimir "2) Calcula la cantidad de artículos en el inventario"
    leer i
    si (i=1)
    inicio
        util← utilidad (a)
        imprimir "La utilidad de la empresa sería:" util
```

(*continúa*)

(*continuación*)

Pseudocódigo

```
        fin
        si no
        inicio
                t_art ← cantidad_total(a)
                imprimir "La cantidad total en el inventario es:", t_art
        fin
fin
```

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
struct articulo
{
    int cod, cant;
    char desc[20]  ;
    float prec_com, prec_ven;
} tlap[500];
float utilidad (int a)
{
    int i;
    float util=0;
    for (i=0; i < a; i++)
        util=util + (tlap[i].prec_ven – tlap[i].prec_com)*tlap[i].cant;
    return until;}
int cantidad_total (int a)
{
    int i, t_art=0;
    for (i=0; i< a; i++)
        t_art=t_art + tlap[i].cant;
    return t_art;
}
main()
{
    int a, i, t_art;
    float util;
    printf ("Total de artículos");
    scanf("%d",&a);
    for (i=0; i<a; i++)
    {
        printf ("Dame los datos\n");
        printf ("Código");
        scanf("%d",&tlap[i].cod);
        printf ("Descripción");
        fflush(stdin);
```

(*continúa*)

(*continuación*)

Lenguaje C

```
        gets(tlap[i].desc);
        printf ("Precio de compra");
        scanf("%f",tlap[i].prec_com);
        printf ("Cantidad en el inventario");
        scanf("%d",&tlap[i].cant);
        tlap[i].prec_ven=tlap[i].prec_com*1.35;
    }
    printf ("1)Calcula la utilidad de la empresa\n");
    printf ("2)Calcula la cantidad de artículos en el inventario\n");
    scanf("%d",&i);
    if (i==1)
    {
        util=utilidad (a);
        printf ("La utilidad de la empresa sería: %f",util);
    }
    else
    {
        t_art=cantidad_total(a);
        printf ("La cantidad total en el inventario es: %d", t_art);
    }
    getch(); return 0;
}
```

Este ejercicio utiliza la estructura artículo, pero a diferencia del ejercicio 3 se declara el arreglo *tlap[500]* de manera global es decir inmediatamente después de declarar la estructura *articulo*. Esto nos ayuda a que pueda ser actualizado o consultado en cualquier parte del programa, es decir en las funciones o en el programa principal. La primera parte del programa es igual para almacenar los datos, el arreglo y calcular el precio de venta. Después se utiliza la variable *i*, para que el usuario identifique qué cálculo desea realizar, para esto se imprime un menú con las dos opciones posibles: ("1) Calcula la utilidad de la empresa", imprimir "2) Calcula la cantidad de artículos en el inventario"). Las dos funciones utilizan al arreglo *tlap* para realizar sus cálculos, sin embargo es necesario enviar por paso de parámetro por valor el "total de artículos" almacenado en *a*; si se elige la opción *i=1* se llama a la función *utilidad (a)* y si elige la *i=0* llama a la función *cantidad_total(a)*.

Ejemplos



Si utilizamos los datos del ejercicio 3 y se llama a la función utilidad (4) y a la función cantidad_total donde *a=4* se realizarán los siguientes cálculos:

prec_com	*cant*	*Prec_ven*	*Prec_ven- prec_com*	*Prec_ven-prec_com *cant*
100	5	135.0	135.0 − 100 = 35	35*5 = 175
130	4	175.5	175.5 − 130 = 45.5	45.5*4 = 182
10	25	13.5	13.5 − 10 = 3.5	3.5*25 = 87.5
50	12	67.5	67.5 − 50 = 17.5	17.5*12 = 210
	46			654.5

Si el usuario elige la opción *i =1* se imprime **654.5** y si elige *i =2* se imprime **46**.

Ejercicios complementarios de registros o estructuras en pseudocódigo

Complete los espacios faltantes en los siguientes programas:

Ejercicio 1. Complete el siguiente ejercicio que almacena los siguientes campos de un estudiante: código, nombre, código de la carrera, semestre, promedio y con base en cuatro materias calcula su promedio y lo almacena en un campo específico.

```
registro info
inicio
     caracter cod_est[9], nom_est[40]
     entero sem_est, cod_carr
     real cal1,cal2,cal3,cal4, prom_est
fin
define_tipo ___________________ tipo_estud
principal()
inicio
___________________ estudiante
     imprimir"Dame los datos"
     imprimir "Código del estudiante"
     leercad _____________________
     imprimir "____________________"
     leercad estudiante.nom_est
     imprimir "¿Qué semestre cursa?"
     leer ____________________
     imprimir "____________________"
     leer estudiante.cod_carr
     imprimir "Dame las cuatro calificaciones"
     ____________________ estudiante.cal1,estudiante.cal2, ____________________ ,estudiante.cal4,
     estudiante.prom← _____________________________________________________
     Imprimir estudiante.prom
fin
```

Ejercicio 2. Utilizando el ejercicio 1 complementario almacene en un arreglo de estructuras los datos de un conjunto de *n* estudiantes y encuentre cuál es el estudiante con menor promedio.

```
registro info
inicio
     caracter cod_est[9], nom_est[40]
     entero sem_est, cod_carr
     real cal1,cal2,cal3,cal4, prom_est
```

(*continúa*)

(continuación)

```
fin
define_tipo __________________ tipo_estud
principal()
inicio
    __________________ estudiante
    entero n, i, est_men; real prom ← 101
    leer n
    desde (i ← 0, __________________ , i ← i + 1)
    inicio
        imprimir"Dame los datos"
        imprimir "Código del estudiante"; leercad __________________
        imprimir "__________________ "; leercad estudiante[i].nom_est
        imprimir "¿Qué semestre cursa?"; leer __________________
        imprimir "__________________ "; leer estudiante[i].cod_carr
        imprimir "Dame cuatro calificaciones";
        ______________ estudiante[i].cal1,estudiante[i].cal2, _______________ , estudiante[i].cal4,
        estudiante[i].prom ← _________________________________________________________________
        imprimir estudiante[i].prom
        si ( ________________________ < prom)
        inicio
            est_menor ← ________________
            prom ← ____________________________
        fin
    fin
    imprimir "El estudiante con menor promedio es",______________ , "y su promedio fue",______________
fin
```

Ejercicio 3. Almacenar en un arreglo de tipo registro o estructura los datos de los libros de una biblioteca (máximo 1000) y hacer una función que nos regrese cuántos libros están prestados.

```
registro datos_libro
inicio
    _______________ cod, cant_inv, cant_prest
    caracter titulo[20] , nombre_autor[40]
fin
define_tipo ____________ total_libros_prestados (libros a[ ],entero n)
inicio
    entero i, cant←0
    desde (i ← 0, i < n, i ← i+1)
        cant ← cant +________________________
    regresa cant
fin
_____________________ datos_libro libros
```

(continúa)

(*continuación*)

```
principal()
inicio
    entero i, t_libros, t_librosprest
    libros biblioteca[100]
    imprimir "Total de títulos de libros (max 1000)"; leer ____________________
    desde (i ← 0, i < t_libros, i ← i+1)
    inicio
        imprimir"Dame los datos del título" i+1
        imprimir "Titulo"; ____________ biblioteca[i].titulo
        imprimir "Autor del libro"; leer biblioteca[i].autor
        imprimir "Código del libro"; leer biblioteca[i].___________
        imprimir "Cantidad de ejemplares"; leer biblioteca[i].cant_inv
        imprimir "Cantidad de ejemplares prestados"; leer __________________cant_prest
    fin
    t_librosprest← total_libros_prestados (___________________) //Llamar a la función
    imprimir "El total de libros prestados es:", ___________________
fin
```

Ejercicios complementarios de registros o estructuras en lenguaje C

Parte I. Complete los espacios faltantes en los siguientes programas:

Ejercicio 1. Con ayuda de un menú capturar y mostrar el nombre, la dirección y la edad de *n* alumnos de un grupo.

```
#include <stdio.h>
____________________
#define MAX 50
____________ estudiante
{
   char nom[35], direc[35];
   _________________
}alumno[MAX];
void capturar(int x)
{
   for (int i=0; i <x; i++)
```

(*continúa*)

(continuación)

```
    {
       printf ("\nNombre: "); fflush(stdin); ______________________
       printf ("\nDomicilio: ");
       fflush(stdin); gets(alumno[i].direc);
       printf ("\nEdad: "); scanf("%d",&alumno[i].edad);
    }
}
__________________(int x, int y)
{
    if (x == 0) printf("NO HAS CAPTURADO DATOS\n");
    else
    {
       if (x< y)
           {
              y =x; printf("El total de datos capturados es: %d\n",x);
           }
       for (int i=0;_________;i ++)
       {
           printf("\nNombre %s\n",______________________);
           printf("Domicilio %s\n",alumno[i].direc);
           __________________________________________
       }
    }
}
main()
{
    int opc,n=0,tmos;
    do
    {
       printf("\n 1.- Capturar \n 2.- Mostrar\n 3.- Salir\n"); scanf("%d",&opc);
       if (opc==1)
           {
                printf ("Dame el total de alumnos a capturar: ");
                scanf ("%d",&n);
                ______________________
           }
       ____________
           {
                printf ("Dame el total de alumnos a mostrar: ");
                scanf ("%d",&tmos);
                mostrar(n,tmos);
           }
    }while (opc!=3); return 0;
}
```

Ejercicio 2. Leer los datos (nombre, edad y sueldo) de *n* empleados (máximo 30) e imprima los trabajadores con sueldo máximo y mínimo, así como la media de los sueldos.

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
struct datos
{
   char nom[30];
   int edad;
   float sueldo;
};
typedef struct _________ trabajador;
main()
{
   trabajador emp[30];
   int i,n,min=0,max=0;
   float media, acum=0;
   printf (“Dame el total de trabajadores: ”);
   scanf(“%d”,&n);
   for(i=0;i<n;i ++)
   {
      printf(“Nombre : “);
      fflush(stdin);
      gets(emp[i].nom);
      printf(“Edad: “);
      scanf(“%d”,&emp[i].edad);
      printf(“Sueldo: “);
      scanf(“%f”,&emp[i].sueldo);
   }
   for(i=1;i < n;i ++)
      if(emp[i].sueldo < emp[min].sueldo)min=i;
   for(i=1;i < n;i++)
      __________________________________
   for(i=0; i <+ n;i ++)
      acum+=(float)emp[i].sueldo;
   media=acum/n;
   printf (“Datos del trabajador con el menor salario\n”);
   printf(“Nombre: “);
   _______________________
   printf(“Edad: ”);
   printf(“%d\n”,emp[min].edad);
   printf(“Sueldo: ”);
   printf(“%f”,emp[min].sueldo);
   printf (“\nDatos del trabajador con el mayor salario\n”);
```

(*continúa*)

(*continuación*)

```
    printf("Nombre : ");
    puts(emp[max].nom);
    printf("Edad: ");
    _______________________
    printf("Sueldo: %f\n",emp[max].sueldo);
    printf("Media: %f\n",media);
    getch();
    return 0;
}
```

Parte II. Elabore la codificación en lenguaje C de los siguientes programas utilizando registros o estructuras (realizando el cálculo e impresión respectiva):

Ejercicio 1. Almacenar en un arreglo los datos personales de los alumnos (máximo 40) de un salón de clases (código, nombre, dirección, edad, sexo, cal1, cal2, promedio), calcular el promedio y almacenarlo en el campo promedio, posteriormente imprimir:

a) Lista de los alumnos.
b) Lista de los alumnos que tienen una determinada edad y aprobaron.
c) Promedio del grupo.
d) Promedio de los alumnos comprendido entre un rango de edades especificado por el usuario.

Ejercicio 2. Almacenar en un arreglo los nombres y las edades de 10 personas. Buscar el nombre correspondiente a una edad determinada y la edad correspondiente a una persona. Ambas opciones deben ser accesibles a través de un menú:

a) Buscar por nombre.
b) Buscar por edad.
c) Salir.

Ejercicio 3. El encargado de un aeropuerto necesita una lista de los vuelos de llegada y de partida ordenados por código de vuelo. Los datos que se tienen de cada vuelo son:

Campo	*Descripción del tipo de dato*
Código de vuelo	Entero
Número de pasajeros en el vuelo	Entero
Línea aérea	Cadena (máximo 10 caracteres)
Hora de llegada	Real (2 decimales, parte entera de hora y parte decimal de minutos) Ejemplo: 18.45
Origen	Cadena de 3 caracteres
Hora de salida	Real (2 decimales) Ejemplo: 10.30
Destino	Cadena de 3 caracteres

Elabore un programa que lea la información de *n* cantidad de vuelos, la ordene en forma descendente con respecto al código de vuelo y la imprima. Considere que como máximo hay 100 vuelos.

Ejercicio 4. Elabore una tabla de proveedores (máximo 30); de cada proveedor se capturará: código, nombre, cantidad vendida del artículo y precio unitario. Calcule el importe. Al final imprimir los datos de cada proveedor, el importe total de compra, el nombre del proveedor más barato y el más caro.

Ejercicio 5. En un videoclub se lleva la siguiente información de cada película: código, título, género y cantidad de préstamos. Capturar la información de las películas que se tienen en existencia (máximo 100), posteriormente imprimir:

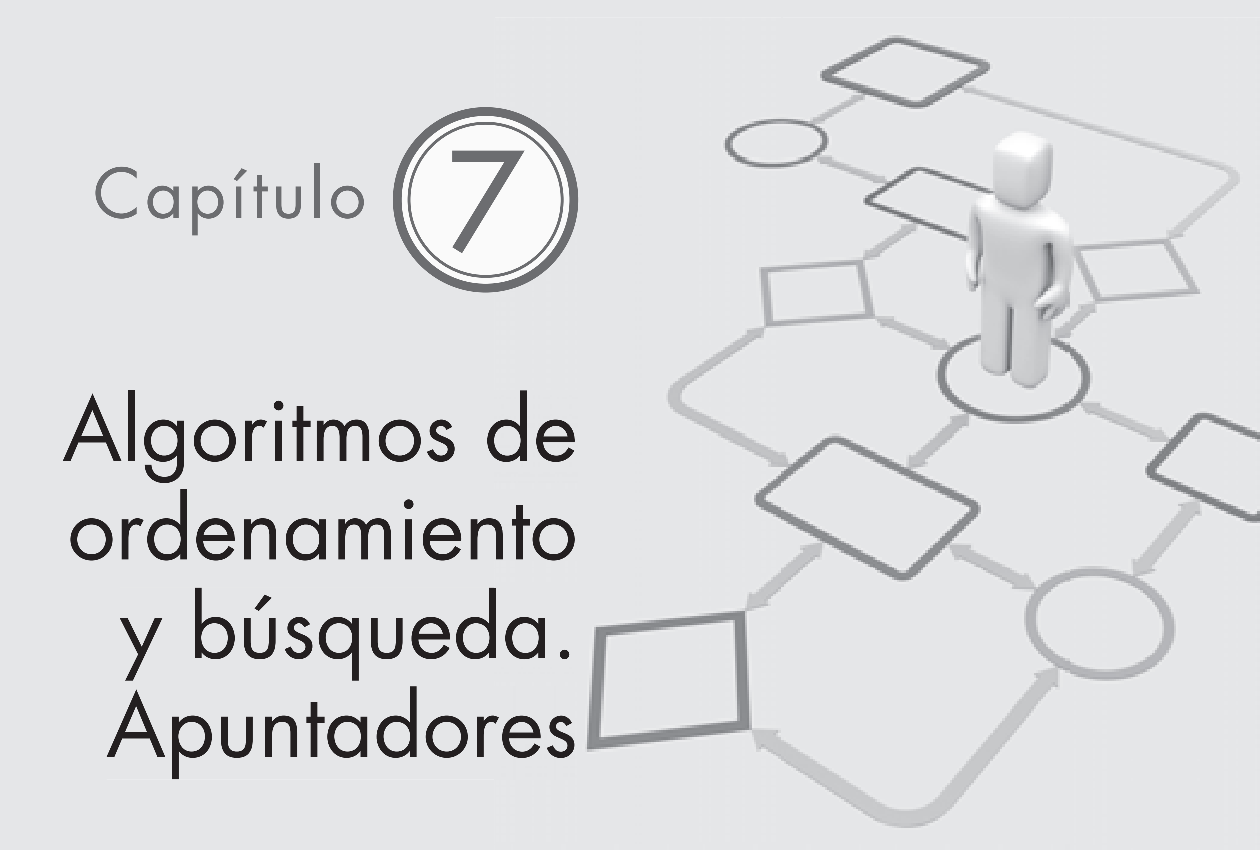
a) Los títulos de un género específico.

b) Las 10 películas más demandadas en orden decreciente.

c) Las 10 películas menos demandadas en orden creciente.

Capítulo 7

Algoritmos de ordenamiento y búsqueda. Apuntadores



Al término de este capítulo, el alumno será capaz de

Objetivo de ordenamientos y búsquedas

- El alumno será capaz de conocer y aplicar los métodos de ordenamiento y de búsqueda más utilizados, para facilitar la manipulación de los datos e identificar el método más adecuado según el tamaño del arreglo o archivo.

Objetivo de apuntadores

- El alumno será capaz de conocer y aplicar el concepto de apuntador para elaborar programas, donde podrá manejar de manera eficiente la memoria.

Contenido

7.1 Ordenación



Su finalidad es clasificar datos (generalmente arreglos o archivos) en un orden ascendente o descendente mediante un criterio (numérico, alfabético...). De acuerdo con el tipo de elemento que se quiera la ordenación puede ser:

a) Interna. Los datos se encuentran en memoria (arreglos, listas, etc.) y pueden ser de acceso aleatorio o directo.

b) Externa. Los datos están en un dispositivo de almacenamiento externo (archivos) y su ordenación es más lenta que la interna.

Los métodos de ordenación interna se suelen dividir en dos grandes grupos:

a) Directos: Burbuja, Selección, Inserción Directa, etcétera.

b) Indirectos (avanzados): Shell, Ordenación Rápida, Ordenación por Mezcla.

En este capítulo sólo nos dedicaremos a la ordenación interna (Burbuja, Selección, Inserción Directa y Shell). Los ejemplos y algoritmos manejan el criterio numérico y ascendente. Además están implementados en lenguaje C.

7.1.1 Ordenación interna

Los métodos de ordenación interna se aplican principalmente a arreglos unidimensionales. A continuación explicaremos sólo los que consideramos más relevantes.

Ordenamiento por Burbuja

Consiste en comparar pares de elementos contiguos e intercambiarlos, comenzando desde la casilla cero del vector hasta haber acomodado el número más grande en la última posición (orden ascendente). Se llama Burbuja porque en cada pasada brota el elemento mayor hasta que queden todos los elementos ordenados. El problema de este algoritmo es que vuelve a comparar los ya ordenados desde la posición cero, por lo que es el más deficiente, sin embargo por su sencillez es el más utilizado.

En la tabla 7.1 el control de las pasadas lo lleva la variable *i*; desde la tercera pasada están ordenados, pero la máquina no lo sabe, por lo tanto tuvimos que realizar la cuarta pasada:

Tabla 7.1 Ejemplo de algoritmo Burbuja

Variables				*Vector*							*Valores ordenados*
				Posición	*0*	*1*	*2*	*3*	*4*	*5*	
i	**j**	**a[j]**	**a[j+1]**	**Original**	**67**	**4**	**30**	**12**	**5**	**27**	**Ninguno**
0	0	67	4	Intercambio	4	67	30	12	5	27	Ninguno
0	1	67	30	Intercambio	4	30	67	12	5	27	Ninguno
0	2	67	12	Intercambio	4	30	12	67	5	27	Ninguno
0	3	67	5	Intercambio	4	30	12	5	67	27	Ninguno
0	4	67	27	Intercambio	4	30	12	5	27	67	67
1	0	4	30	Igual	4	30	12	5	27	**67**	67
1	1	30	12	Intercambio	4	12	30	5	27	**67**	67
1	2	30	5	Intercambio	4	12	5	30	27	**67**	67
1	3	30	27	Intercambio	4	12	5	27	**30**	**67**	67, 30

(*continúa*)

Tabla 7.1 Ejemplo de algoritmo Burbuja (*continuación*)

Variables				*Vector*							*Valores ordenados*
				Posición	***0***	***1***	***2***	***3***	***4***	***5***	
2	0	4	12	Igual	4	12	5	27	**30**	**67**	67, 30
2	1	12	5	Intercambio	4	5	12	27	**30**	**67**	67, 30
2	2	12	27	Igual	4	5	12	**27**	**30**	**67**	67, 30, 27
3	0	4	5	Igual	4	5	12	**27**	**30**	**67**	67, 30, 27
3	1	5	12	Igual	4	5	12	**27**	**30**	**67**	67, 30, 27, 12
4	0	4	5	Igual	4	5	12	**27**	**30**	**67**	67, 30, 27, 12.5
				Ordenado	4	5	12	27	30	67	

Lenguaje C de Algoritmo Burbuja

```
#include <stdio.h>
#include <conio.h>
# define TAM 6
main()
{
   int vec[TAM]={67,4,30,12,5,27},i,j,aux;
   printf("ARREGLO ORIGINAL\n");
   for (i=0;i<TAM;i++)
       printf("%4d",vec[i]); printf("\n");
   for(i=0; i < TAM-1; i++)
   {
       for(j=0; j < TAM-1; j++)
       {
          if(vec[j] > vec[j+1])
             {
                aux=vec[j];
                vec[j]=vec[j + 1];
                vec[j+ 1]=aux;
             }
       }
       printf("PASADA %d: ",i+1);
       for(j= 0;j<TAM;j++)
          printf("%d ",vec[j]);
       printf("\n");
   }
   printf("ARREGLO ORDENADO\n");
   for (i= 0;i<TAM;i++)
       printf("%4d",vec[i]); getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

ARREGLO ORIGINAL
67 4 30 12 5 27
PASADA 1: 4 30 12 5 27 67
PASADA 2: 4 12 5 27 30 67
PASADA 3: 4 5 12 27 30 67
PASADA 4: 4 5 12 27 30 67
PASADA 5: 4 5 12 27 30 67
ARREGLO ORDENADO
4 5 12 27 30 67

En la versión mejorada de método de la Burbuja nos auxiliamos de una bandera, la cual en el momento en que su valor sigue siendo 1, nos indica que los elementos ya fueron clasificados y por lo tanto se sale del ciclo *while* y termina. El programa es el siguiente:

Lenguaje C de Algoritmo Burbuja Mejorado

```
#include <stdio.h>
#include <conio.h>
# define TAM 6
main()
{
    int vec[TAM]={67,4,30,12,5,27},i,j,aux,band=0;
    printf("ARREGLO ORIGINAL\n");
    for (i=0;i< TAM;i++)
         printf("%4d",vec[i]);
    printf("\n");
    i=1;
    while (i<TAM && band==0)
    {
         band=1;
         for(j= 0; j< TAM-i; j++)
         {
             if(vec[j] > vec[j+1])
                 {
                     band= 0;
                     aux=vec[j];
                     vec[j]=vec[j+1];
                     vec[j+1]=aux;
                 }
         } i++;
         printf("PASADA %d: ",i-1);
         for(j=0;j<TAM;j++)
             printf("%d ",vec[j]);
         printf("\n");
    }
    printf("ARREGLO ORDENADO\n");
    for (i=0;i< TAM;i++)
         printf("%4d",vec[i]);
    getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

ARREGLO ORIGINAL
67 4 30 12 5 27
PASADA 1: 4 30 12 5 27 67
PASADA 2: 4 12 5 27 30 67
PASADA 3: 4 5 12 27 30 67
PASADA 4: 4 5 12 27 30 67

ARREGLO ORDENADO
4 5 12 27 30 67

Ordenamiento por Selección

Se busca el elemento menor del vector, se intercambia con el de la primera posición; después, se busca el segundo elemento más pequeño y se intercambia con el de la segunda posición y así sucesivamente hasta que todos queden ordenados. En la tabla 7.2 vemos un ejemplo:

Ejemplos

Tabla 7.2 Ejemplo de algoritmo Selección

Variables	Vector						
	Posición	*0*	*1*	*2*	*3*	*4*	*5*
i	Original	67	4	30	12	5	27
0	El 4 va a la primera posición y se intercambia por el 67.	4	67	30	12	5	27
1	El 5 va a la segunda posición y se intercambia por el 67.	4	5	30	12	67	27
2	El 12 va a la tercera posición y se intercambia por el 30.	4	5	12	30	67	27
3	El 27 va a la cuarta posición y se intercambia por el 30.	4	5	12	27	67	30
4	El 30 va a la quinta posición y se intercambia por el 67.	4	5	12	27	30	67
	Ordenado	4	5	12	27	30	67

El programa es el siguiente:

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
#define TAM 6
main()
{
   int vec[TAM]={67,4,30,12,5,27},i,j,aux,men=0;
   printf("ARREGLO ORIGINAL\n");
   for (i=0;i< TAM;i++)
       printf("%4d",vec[i]); printf("\n");
   for(i=0;i< TAM;i++)
   {
       for(j=i+1;j< TAM;j++)
          if(vec[j] < vec[men]) men=j;
       aux = vec[i];
       vec[i] = vec[men];
       vec[men] = aux;
       men = i + 1;
       for(j=0;j< TAM;j++)
          printf("%d ",vec[j]);
       printf("\n");
   }
   printf("\nARREGLO ORDENADO\n ");
   for(i=0;i< TAM;i++)
       printf("%3d",vec[i]);
   getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

```
ARREGLO ORIGINAL
67 4 30 12 5 27
4 67 30 12 5 27
4 5 30 12 67 27
4 5 12 30 67 27
4 5 12 27 67 30
4 5 12 27 30 67
4 5 12 27 30 67

ARREGLO ORDENADO
4 5 12 27 30 67
```

Ordenamiento por Inserción Directa

Se va insertando el nuevo elemento de tal manera que la lista generada vaya quedando ordenada. El método de ordenación por Inserción Directa lo utilizan generalmente los jugadores de cartas, de ahí que también se conozca con el nombre de método de la baraja. En la tabla 7.3 vemos un ejemplo:

Ejemplos



Tabla 7.3 Ejemplo de algoritmo de Inserción Directa

Variables					*Vector*							*Valores ordenados*
					Posición	*0*	*1*	*2*	*3*	*4*	*5*	
i	**j=i−1**	**aux=vec[i]**	**vec[j+1]=vec[j]**	**vec[j+1]=aux**	**Original**	**67**	**4**	**30**	**12**	**5**	**27**	**Ninguno**
1	0	aux=4	vec[1]=67			67	67	30	12	5	27	Ninguno
1	−1	Sale del ciclo while		vec[0]=4	Intercambio	4	67	30	12	5	27	4
2	1	30	vec[2]=67			4	67	67	12	5	27	4
2	0	Sale del ciclo while		vec[1]=30	Intercambio	4	30	67	12	5	27	4
3	2	aux=12	vec[3]=67			4	30	67	67	5	27	4
3	1		vec[2]=30			4	30	30	67	5	27	4
3	0	Sale del ciclo while		vec[1]=12	Intercambio	4	12	30	67	5	27	4
4	3	aux=5	vec[4]=67			4	12	30	67	67	27	4
4	2		vec[3]=30			4	12	30	30	67	27	4
4	1		vec[2]=12			4	12	12	30	67	27	4
4	0	Sale del ciclo while		vec[1]=5	Intercambio	4	5	12	30	67	27	4, 5
5	4	aux=27	vec[5]=67			4	5	12	30	67	27	4, 5
5	3		vec[4]=30			4	5	12	30	30	67	4, 5
5	2	Sale del ciclo while		vec[3]=27	Intercambio	4	5	12	27	30	67	4, 5, 12, 27, 30, 67
					Ordenado	4	5	12	27	30	67	

Lenguaje C

```
#include <stdio.h>
#include <conio.h>
# define TAM 6
main()
{
    int vec[TAM]={67,4,30,12,5,27},i,j,aux;
    printf("ARREGLO ORIGINAL\n");
    for (i=0;i<TAM;i++)
```

(continúa)

(*continuación*)

```
        printf("%4d",vec[i]);
    printf("\n");
    for (i=1;i<TAM;i++)
    {
        j=i-1;
        aux=vec[i];
        while (aux<vec[j]&&j>=0)
        {
              vec[j+1]=vec[j];
              j--;
        }
        vec[j+1]=aux;
        for(j=0;j<TAM;j++)
        printf("%d ",vec[j]);
        printf("\n");
    }
    printf("ARREGLO ORDENADO\n");
    for (i=0;i< TAM;i++)
        printf("%4d",vec[i]);
    getch();
    return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

```
ARREGLO ORIGINAL
67  4 30 12  5 27
 4 67 30 12  5 27
 4 30 67 12  5 27
 4 12 30 67  5 27
 4  5 12 30 67 27
 4  5 12 27 30 67

ARREGLO ORDENADO
 4  5 12 27 30 67
```

Ordenamiento Shell

Es una mejora del método de inserción directa, se utiliza cuando el número de elementos a ordenar es muy grande. En este método no se compara a cada elemento con el de su izquierda, como en el de inserción, sino con el que está a un cierto número de lugares (llamado salto) a su izquierda. Este salto es constante y su valor inicial es N/2 (N es el número de elementos y división entera). Se van dando pasadas hasta que en una pasada no se intercambie ningún elemento de sitio. Entonces el salto se reduce a la mitad, y se vuelven a dar pasadas hasta que no se intercambie ningún elemento, y así sucesivamente hasta que el salto valga 1.[1] En la tabla 7.4 vemos el ejemplo para ordenar el arreglo [6, 1, 5, 2, 3, 4, 0]





Tabla 7.4 Ejemplo de algoritmo Shell

Recorrido	*Salto*	*Lista ordenada*	*Intercambio*
1	3	2, 1, 4, 0, 3, 5, 6	(6,2), (5,4), (6,0)
2	3	0, 1, 4, 2, 3, 5, 6	(2,0)
3	3	0, 1, 4, 2, 3, 5, 6	Ninguno
4	1	0, 1, 2, 3, 4, 5, 6	(4,2), (4,3)
5	1	0, 1, 2, 3, 4, 5, 6	Ninguno

[1] http://www.algoritmia.net/articles.php?id=31.

Lenguaje C

```
#include<stdio.h>
#include <conio.h>
#define MAX 7
main()
{
   int vec[MAX]={6,1,5,2,3,4,0}, salto, aux, i, j, cambio;
   printf("ARREGLO ORIGINAL\n");
   for (i=0;i< MAX;i++)
       printf("%4d",vec[i]);
   printf("\n");
   salto=MAX/2;
   while (salto>0)
   {
       cambio=1;
       while (cambio!=0)
       {
            cambio=0;
            for(i=salto;i< MAX;i++)
            {
                j=i−salto;
                if(vec[j]> vec[i])
                {
                    aux=vec[i];
                    vec[i]=vec[j];
                    vec[j]=aux;
                    cambio++;
                    printf("Intercambio: %d,%d\n", vec[i],vec[j]);
                }
            }
            for (i=0; i< MAX; i++)
                printf("%d ", vec[i]);
            printf("\n");
       }
       salto/= 2;
   }
   printf("ARREGLO ORDENADO\n");
   for (i=0;i< MAX;i++)
       printf("%4d",vec[i]);
   getch(); return 0;
}
```

Una posible salida en pantalla, después de ejecutar el programa sería:

```
ARREGLO ORIGINAL
6 1 5 2 3 4 0
Intercambio: 6,2
Intercambio: 5,4
Intercambio: 6,0
2 1 4 0 3 5 6
Intercambio: 2,0
0 1 4 2 3 5 6
0 1 4 2 3 5 6
Intercambio: 4,2
Intercambio: 4,3
0 1 2 3 4 5 6
0 1 2 3 4 5 6
ARREGLO ORDENADO
0 1 2 3 4 5 6
```

Ejercicios complementarios de ordenamiento

Ejercicio 1. Anote en el recuadro de la derecha el método utilizado para ordenar en cada caso, dada la siguiente secuencia de elementos.

Original	8	5	6	4	7	3

Paso	Valores					
1)	5	8	6	4	7	3
2)	5	6	8	4	7	3
3)	5	6	4	8	7	3
4)	5	6	4	7	8	3
5)	5	6	4	7	3	8
6)	5	4	6	7	3	8
7)	5	4	6	3	7	8
8)	4	5	6	3	7	8
9)	4	5	3	6	7	8
10)	4	3	5	6	7	8
11)	3	4	5	6	7	8

Ordenación por ______________________

Paso	Valores					
1)	3	5	6	4	7	8
2)	3	4	6	5	7	8
3)	3	4	5	6	7	8
4)	3	4	5	6	7	8
5)	3	4	5	6	7	8

Ordenación por ______________________

Paso	Valores					
1)	5	8	6	4	7	3
2)	5	6	8	4	7	3
3)	4	5	6	8	7	3
4)	4	5	6	7	8	3
5)	3	4	5	6	7	8

Ordenación por ______________________

(*continúa*)

(*continuación*)

Paso	Valores					
1)	4	5	6	8	7	3
2)	4	5	3	8	7	6
3)	4	3	5	8	7	6
4)	4	3	5	7	8	6
5)	4	3	5	7	6	8
6)	3	4	5	7	6	8
7)	3	4	5	6	7	8

Ordenación por ______________________________

Ejercicio 2. Complete las siguientes líneas:

1. El método de ordenamiento ____________________ encuentra el elemento mayor o menor de la lista, intercambie este elemento con el elemento que tiene la posición adecuada (si es el primer elemento a buscar se coloca en la primera posición). A continuación se busca el siguiente elemento en la lista y se intercambia colocándolo en la siguiente posición y así sucesivamente hasta que quede ordenado el arreglo.
2. El método de ordenamiento ______________________ se suele denominar también ordenación por disminución de incremento.

7.2 Búsquedas

7.2.1 Búsqueda secuencial

La búsqueda lineal o secuencial es la manera más sencilla de buscar un elemento en un vector. Se recorre desde el primer elemento hasta el último, de uno en uno. Si se encuentra el elemento, el resultado será la posición del elemento buscado y si no aparecerá el mensaje respectivo o cero (es falso que se encontró).

Ejemplos



Si el número que se busca es bus=12 y el arreglo respectivo es: [67, 4, 30, 12, 5, 27], con ayuda de un ciclo *for* se irá comparando el valor de bus=12 con cada elemento del vector *vec[i]*, donde *i* varía de 0 a 5. Si se encuentra se almacenará la posición del vector en una variable; es usual que el resultado de la citada variable se regrese al programa principal, si es que éste hizo la llamada. El programa es el siguiente:

Lenguaje C

```
#include<conio.h>
#include<stdio.h>
#define MAX 45
main()
{
    int vector[MAX],i,n,ban=0,x;
    printf("Dame el número de valores a ingresar \n"); scanf("%d",&n);
    for(i=0;i< n;i++)
    {
```

(*continúa*)

(*continuación*)

Lenguaje C

```
        printf("Dame los valores a ingresar %d",i+1);
        scanf("%d", & vector [i]);
    }
    printf("Dame el número a buscar "); scanf("%d",&x);
    for(i=0;i< n;i++)
    if(vector[i]==x)
    {
        printf("Lo encontré en la posición %d\n",i+1);
        ban=1;
    }
    if(ban==0)printf ("No lo encontré ");
    getch(); return 0;
}
```

cuando sólo queremos conocer la primera aparición del elemento en el vector, tenemos:

```
for(i=0;i<n;i++)
    if(vector[i]==x)
         break;
```

Este programa tiene un inconveniente, sea cual sea el resultado se recorre el vector completo. El programa se puede mejorar con un bucle *while* o *do_while*. El bucle se terminará por dos causas:

1. El valor fue encontrado (*x==vector[i]*).
2. El valor de la variable que recorre el vector *(i)* es mayor que el número de elementos de la lista (MAX), lo que significa que se ha terminado de recorrer la misma y el elemento buscado no lo ha encontrado.

Lenguaje C

```
#include<conio.h>
#include<stdio.h>
#define MAX 5
main ()
{
    float vector[MAX];
    int i,x;
    for (i = 0; i < MAX; i=i+1)
    {
        printf ( "Dame el vector de datos %d: " ,i+1);
        scanf ("%f",&vector[i]);
    }
    printf("Dame el número a buscar "); scanf ("%d",&x);
    i=0;
    while (i< MAX && x!= vector[i])
        i++;
```

(*continúa*)

(*continuación*)
```
    if (i!= MAX)
            printf("Lo encontré en la posición %d\n",i+1);
        else printf ("No lo encontré ");
    getch(); return 0;
}
```

En este último programa, sólo nos interesa la primera vez que aparece. Otra posibilidad es implementar una función que haga la búsqueda correspondiente:

```
int busqueda_sec(int v[], int n, int nbus)
{
    int i=0;
    while(i< n && a[i]!= nbus)
                i++;
    if(i < n)
            return(i);
        else
            return(-1);
}
```

7.2.2 Búsqueda binaria

La búsqueda secuencial no es efectiva cuando el vector es muy grande y el elemento que se busca está al final del arreglo. La búsqueda binaria se basa en el algoritmo divide y vencerás. Para poder utilizar la búsqueda binaria el vector debe estar ordenado. Primeramente se divide el arreglo a la mitad, comparando el elemento central con el número a buscar, si es igual el valor fue encontrado; en caso contrario si el elemento a buscar es menor se realiza la búsqueda en los elementos de la izquierda y si es mayor al elemento medio se busca a la derecha del mismo.

Ejemplos



Para buscar el elemento 7 en el vector {1,2,4,5,7,9,11,16,21,25,34} seguimos estos pasos:

1. Dividimos el arreglo entre 2, como el tamaño es 11 al dividirlo obtenemos 5, por lo que el elemento central es el 9, lo cual genera dos subgrupos {1,2,4,5,7} y {11,16,21,25,34}.
2. Como el valor a buscar es menor al central ($7 < 9$), se toma el subgrupo de la izquierda y se vuelve a dividir entre dos, ahora el elemento medio es el 4 y los subgrupos {1,2} y {5,7}; tomamos el subgrupo de la derecha. El programa es el siguiente:

Lenguaje C

```
#include<conio.h>
#include<stdio.h>
#define TAM 11
int busq_bin(int v[], int n, int nbus)
{
    int izq=0, der=n-1, medio;
    while(izq <= der)
    {
        medio = (izq + der) / 2;
```

(*continúa*)

(*continuación*)

```
Lenguaje C
         if (v[medio] == nbus) return (medio);
             else if (v[medio] < nbus) izq = medio + 1;
                  else der = medio − 1;
    }
    return(−1);
}
main ()
{
    int vec[TAM]={1,2,4,5,7,9,11,16,21,25,34},i,x;
    printf("ARREGLO ORIGINAL\n");
    for (i=0;i< TAM;i++)
    printf("%4d",vec[i]);
    printf("\n");
    printf ( "Qué número deseas buscar: " );
    scanf ("%d",&x);
    if (busq_bin(vec,TAM,x)>=0)
             printf("Lo encontré en la posición %d\n",busq_bin(vec,TAM,x));
         else printf ("No lo encontré ");
    getch(); return 0;
}
```

Por las características de la búsqueda binaria podríamos implementar los siguientes módulos:

1. Leer el vector.
2. Ordenar el vector.
3. Búsqueda binaria.

El resultado se podría imprimir en la función *main*.

Ejercicios complementarios de búsquedas

Complete las siguientes líneas:

1. La búsqueda se basa en el conocido método divide y vencerás: ______________________.
2. Búsqueda donde se requiere que los elementos estén ordenados: ____________________.

7.3 Apuntadores



La memoria de una computadora puede verse como un vector de valores, donde cada una de sus posiciones es referenciada por un número hexadecimal llamado Dirección.

Como ya se había comentado en el tema de los datos y operaciones básicas en el capítulo 2, cuando se declara una variable, el compilador reserva un espacio de memoria para ella y asocia el nombre de ésta a la dirección de memoria.

Los apuntadores o punteros son variables cuyo contenido es una dirección de memoria. Estos referencian direcciones de memoria de otras variables (*int*, *float*, *char*). Un apuntador "apunta" a la variable cuyo valor se almacena a partir de la dirección de memoria que contiene el apuntador.

7.3.1 Dirección de una variable

La obtención de la dirección de una variable en C y C++ se lleva a cabo a través del operador unario '&', aplicado a la variable a la cual se desea saber su dirección.

Ejemplos



```
int a = 27;
printf ("La dirección de a es: %x",&a);
```

Se imprimirá un valor hexadecimal (formato %x) como por ejemplo: "0x22ff74". Este valor puede variar durante cada ejecución del programa, ya que el programa puede reservar distintos espacios de memoria durante cada ejecución.

7.3.2 Declaración de apuntadores

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador. La sintaxis es la siguiente:

<tipo_apuntador> *<Identificador_variable>

Indica que la variable que se declara contendrá la dirección de una variable que contiene información del tipo <tipo_apuntador>.

Ejemplos



```
int *p_entero;                 //Apuntador a un dato de tipo entero (int)
char *c1;                      //Apuntador a un dato de tipo carácter (char)
float *p_real1, *p_real2;      //Dos apuntadores a datos de tipo punto flotante (float)
```

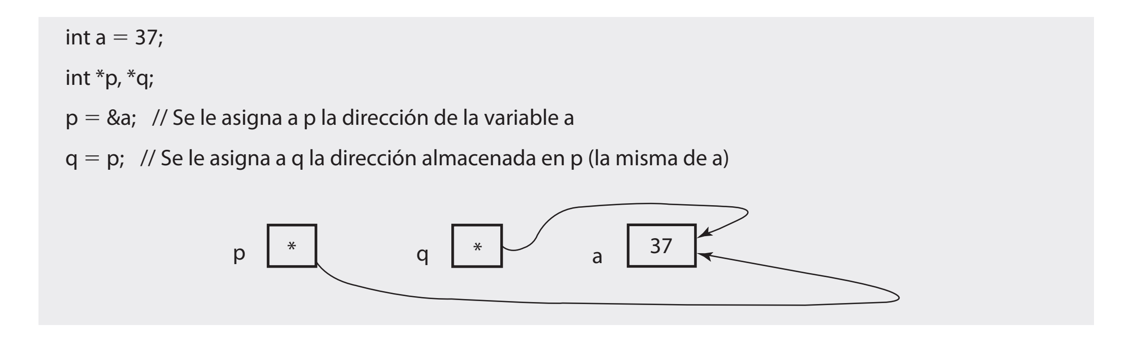
7.3.3 Asignación de apuntadores

Se pueden asignar a un apuntador direcciones de variables a través del operador unario '&' o direcciones almacenadas en otros apuntadores.

Ejemplos



```
int a = 37;
int *p, *q;
p = &a;   // Se le asigna a p la dirección de la variable a
q = p;   // Se le asigna a q la dirección almacenada en p (la misma de a)
```



Para la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador utilizamos el operador unario '*', aplicado al apuntador que contiene la dirección del valor.

Ejemplos



```
int a = 37, b;
int *p;
p = &a;
printf ("El valor de a es: %d",*p); //Se imprime 37
b = *p * 2;
printf ("El valor de b es: %d",b); //Se imprime 74
```

7.3.4 El operador −>

Con el operador binario '−>', obtenemos los campos de un registro con un apuntador al mismo. Muchos compiladores de C utilizan este operador.

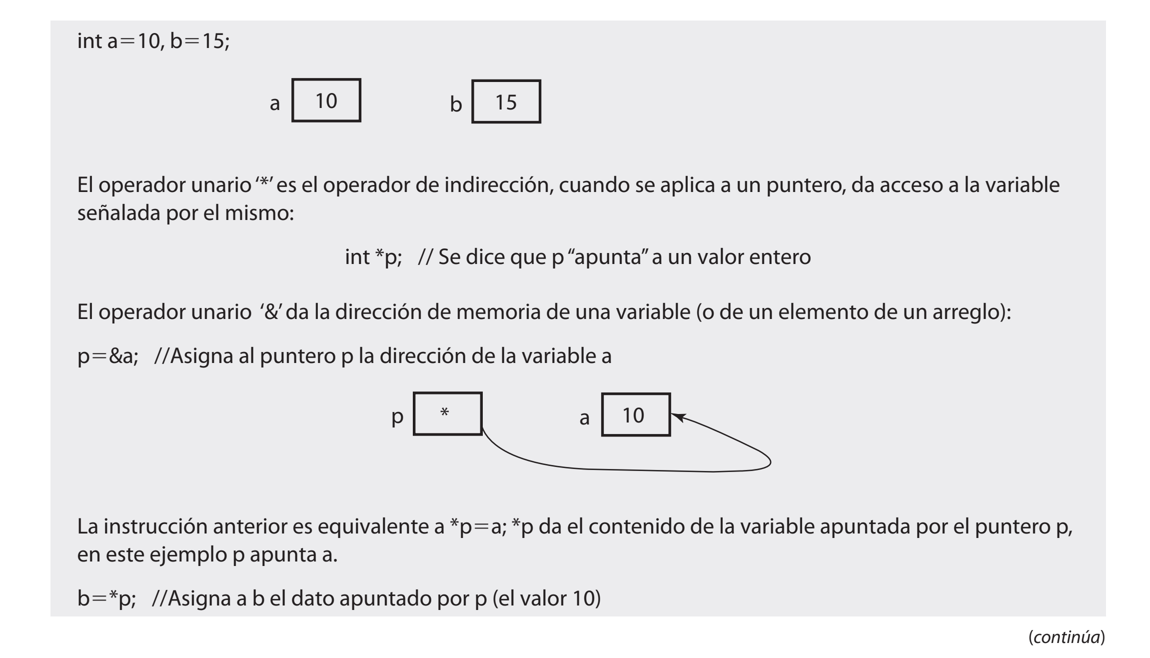
Ejemplos



```
struct empleado
{
        char nombre[35];
        int edad;
};
empleado dato;
empleado *pdato = &dato;
(*pdato).edad =19;   //Asignación del campo 'edad' utilizando el operador .
pdato −> edad = 19;   //Asignación del campo 'edad' utilizando el operador '−>'
printf("%d",pdato −> edad);
```

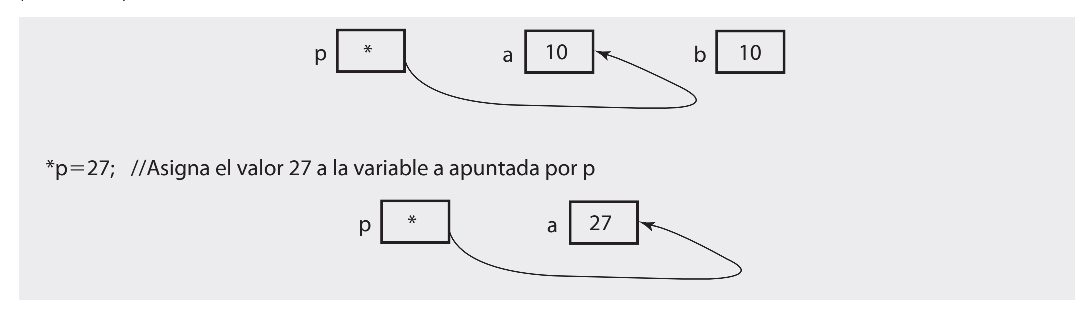
Para reafirmar los conceptos aprendidos veamos los siguientes ejemplos. Comenzamos inicializando las variables a y b:

Ejemplos



(continúa)

(*continuación*)



7.3.5 Errores usuales

```
float *p;
int a;
p = &a;   //El apuntador apunta a un float y la variable es tipo int
float *p;
*p = 3.5;   //p no ha tomado la dirección de variable alguna, por lo tanto al apuntador NO se le puede asignar valor alguno
```

7.3.6 Apuntadores a apuntadores

Dado que un apuntador es una variable que apunta a otra, pueden existir apuntadores a apuntadores, y a su vez los segundos pueden apuntar a apuntadores, y así sucesivamente. Estos apuntadores se declaran colocando tantos asteriscos (‘*’) como sea necesario.

Ejemplos



```
char caracter = 'a';
char *pcaracter = &caracter;
char **ppcaracter = &pcaracter;
char ***pppcaracter = &ppcaracter;
***pppcaracter = 'm'; // Cambia el valor de carácter a 'm'
```

7.3.7 Apuntadores para paso de parámetros por referencia

El lenguaje C no maneja directamente parámetros por referencia. Sin embargo, es posible hacerlo a través del uso de apuntadores. Veamos un ejemplo utilizando los operadores unarios ‘&’ y ‘*’:

```
#include<conio.h>
#include<stdio.h>
void suma(int a, int b, int &c)
{
   c = a + b;
}
void division(float a, float b, float *c)
{
   *c =  a / b;
```

La salida en pantalla, después de ejecutar el programa sería:

10 + 4 = 14

10 / 4 = 2.500000

(*continúa*)

(*continuación*)

```
}
main ()
{
    int res;
    float res1;
    suma(10, 4, res);
    division(10, 4, &res1);
    printf ("10 + 4 =  %d\n",res);
    printf ("10 / 4 = %f",res1);
    getch();
    return 0;
}
```

El operador '&' tiene dos significados como operador unario: indicación de parámetro por referencia y operador de referencia a memoria. En este ejemplo las funciones suma y división tienen los parámetros valor a y b, esto quiere decir que no cambian su valor dentro de la función, y el parámetro c, que es de referencia. Esto quiere decir que los valores de c pueden ser modificados dentro de la función, los cuales están representado como &c en la función suma y como *c en la función división.

El siguiente ejercicio nos muestra cómo pueden cambiar los valores de dos variables de tipo entero y de una variable de tipo apuntador entera.

```
#include <stdio.h>
#include <conio.h>
int x=7, y=5;
int *apunt;
main()
{
    printf( "x=%d y=%d *apunt = %d\n" , x, y, apunt);
    apunt = &x;
    y=*apunt;
    *apunt=25;
    printf("las variables cambiaron y ahora tenemos:\n");
    printf("x=%d y=%d *apunt = %d" , x, y, *apunt);
    getch();
    return 0;
}
```

La salida en pantalla, después de ejecutar el programa sería:

x=7 y=5 *apunt = 0

las variables cambiaron y ahora tenemos:

x=25 y= 7 *apunt = 25

ya que **apunt* apunta a *x*, y se queda con el valor de *x* y a *x* se le cambia su valor por 25.

Ejercicios complementarios de apuntadores

I. Realice la prueba de escritorio o corrida a mano del siguiente programa:

```
#include <stdio.h>
#include <conio.h>
void func1(int *a, int b);
void func2(int a, int &b, int &c);
void func3(int *a);
main()
{
    int x=4,y=2,z=3;
    func2(x,y,z);
    printf("%d %d %d",x,y,z);
    getch(); return 0;
}
void func1(int *a, int b)
{
    *a=*a+3*b;
    b= b+*a;
    printf("%d %d\n",*a,b);
}
void func2(int a, int &b, int &c)
{
    int w=10;
    func1(&a,b);
    printf("%d %d\n",a,b);
    b=a*w; c=c+a;
    printf("%d %d %d\n",a,b,c);
    func3(&c);
}
void func3(int *c)
{
    (*c)++;
}
```

II. Elabore la codificación en lenguaje C de los siguientes programas utilizando apuntadores (realizando el cálculo e impresión respectiva):

1. Con ayuda de un menú elabore un programa con cuatro funciones tipo *void* donde calcule:
   a) Suma.
   b) Resta.

c) Multiplicación.

d) División.

Cada función tendrá tres parámetros, dos por valor (los dos números que recibe) y un parámetro por referencia (el resultado), ya que este último se calculará dentro de la función respectiva. Los datos de entrada y las impresiones de los resultados respectivos se llevarán a cabo en la función *main()*.

2. Elabore una función tipo *void* que reciba una matriz cuadrada de tamaño *n* (máximo 10) y devuelva la suma de las filas pares y la suma de las filas impares. Los resultados se imprimirán en la función *main()*.
3. Elabore una función tipo *void* que reciba una cadena e imprima la cantidad de palabras que tiene una frase en la función *main()*.
4. Elabore una función tipo *void* que reciba una palabra e imprima la letra que se repita más veces en la función *main()*.
5. Cree una función tipo *void* que intercambie el valor de los dos números enteros que se le indiquen como parámetro, los números ya intercambiados se imprimirán en la función *main()*. Para ello se pasarán como parámetros las direcciones de las variables.



Anexo A

Entrada y salida (e/s) en lenguaje C

A.1 Entrada y salida formateada



El lenguaje C no tiene palabras reservadas para la e/s estándar del sistema (teclado y pantalla). Estas operaciones se realizan mediante funciones de biblioteca que encontramos en el archivo de cabecera *stdio.h* (standard input-output header). La e/s con *formato* se realiza por medio de las funciones *scanf()* y *printf*. Las *f* de *scanf() y de printf()* significan "con formato".

A.1.1 Entrada de datos: función *scanf()*

La función *scanf()* se usa para la entrada y nos permite leer (introducir) datos desde el teclado y asignarlos a variables de C, de acuerdo con el formato especificado.

Sintaxis: *scanf ("cadena de control", argumento1,argumento2 …);*

La *cadena de control* tiene uno o varios *especificadores* o *códigos de formato* que harán referencia al tipo de dato de los argumentos y, si se desea, su anchura. Los códigos estarán precedidos por el carácter % seguido de un *carácter de conversión*; los más utilizados y su significado se muestran en la tabla A.1. La *cadena de control* deberá escribirse entre comillas (" ").

La lista de *argumentos* serán las variables; *scanf* necesita saber la posición de memoria de la computadora en que se encuentra la variable para poder almacenar la información obtenida, para ello utilizaremos el símbolo *ampersand (&)*, que colocaremos antes del nombre de cada variable. Por lo tanto, el & sirve para apuntar a la dirección de memoria de dicha variable. A continuación presentamos varios ejemplos.

Ejemplos

```
char x;
scanf ("%c", &x);   //Se captura un carácter y lo almacena en la variable x.
```

No es necesario el & cuando el dato que se va a capturar es una cadena de caracteres, ya que con el nombre de una cadena (en general, de un arreglo) sin índices se identifica la dirección del primer elemento.

Veamos un ejemplo de cadena:

Ejemplos



```
char cad[50];
scanf ("%s", cad);        o        scanf ("%s", &cad);
```

El *scanf ("%f %d %c", &x, &y, &z)*; almacena en *x, y* y *z*, respectivamente, los datos introducidos de cualquiera de las siguientes formas:

- Oprimir la tecla <enter> después de introducir cada dato.
- Oprimir la tecla <TAB> después de introducir cada dato.
- Oprimir un espacio en blanco después de introducir cada dato.
- Alternar cualquiera de las opciones anteriores entre cada dato.

Hay que tener precaución con el uso de los espacios en blanco. Una sentencia como scanf (*"%s ", cadena*); no volverá hasta que no se teclee un carácter no blanco, puesto que el espacio que hay después de *%s* hace que se lean y descarten espacios, tabuladores y saltos de línea.

Otra posibilidad es delimitar la entrada de cada variable, por ejemplo la coma:

Ejemplo

```
scanf("%d,%d", &x, &y); //La entrada podría ser: 15,24
```

Tabla A.1 Tipos de datos y formatos en lenguaje C

Tipo de argumento	*Nombre del dato o formato de salida*	*Especificador o código de formato*	*Bits*	*Rango*
Carácter	char	%c	8	−128 a 127

(*continúa*)

Tabla A.1 Tipos de datos y formatos en lenguaje C (*continuación*)

Carácter sin signo	unsigned char	%u	8	0 a 255
Carácter con signo	signed char	%c	8	−128 a 127
Carácter	Cadena de caracteres terminada en '\0'	%s	Tamaño de la cadena *8+8	
Carácter	Imprime el carácter %	%%	−	−
Entero decimal	int	%d, %i	16 (Turbo C) 32 (Dev−C++ y Code::Blocks)	−32768 a 32767 (2 bytes) −2147483648 a 2147483647
Entero decimal sin signo	unsigned int	%u	16 (Turbo C) 32 (Dev−C++ y Code::Blocks)	0 a 65535 0 a 4294967295
Entero corto decimal	short int	%hd	16	−32768 a 32767
Entero corto decimal sin signo	unsigned short int	%hu	16	0 a 65535
Entero corto decimal con signo	signed short int	%d	16	−32768 a 32767
Entero largo decimal	long int	%ld	32	−2147483648 a 2147483647
Entero largo decimal sin signo	unsigned long int	%lu	32	0 a 4294967295
Entero largo decimal con signo	signed long int	%l	32	−2147483648 a 2147483647
Entero octal	Salida octal sin signo	%o		
Entero hexadecimal sin signo (a - f)	Salida hexadecimal en minúsculas	%x		
Entero hexadecimal sin signo (A - F)	Salida hexadecimal en mayúsculas	%X		
Real (punto flotante)	float	%f, %g, %G, %e, %E	32	3.4E−38 a 3.4E+38
Real	double	%lf, %lg, %lG, %le, %lE	64	1.7E−308 a 1.7E+308
Real	long double	%lf, %lg, %lG, %e, %lE	80 (Turbo C) 96 (Dev−C++ y Code::Blocks)	3.37E−4932 a 1.18E+4932
Puntero		%p		

Notas:
La l previa a los tipos long, double y long double es obligada en el scanf y opcional en el printf.
%g El más corto de %f y %e   %G El más corto de %f y %E
%e Notación científica en minúscula   %E Notación científica en mayúscula

Debemos tener cuidado en NO usar la cadena de control de *scanf()* como en *printf():*

scanf ("Dame un número Entero: %d", &x);

Si no tecleamos el letrero idéntico, el compilador le asignará cualquier valor a la variable x.

Cuando se especifica el carácter *, el dato de entrada no será tomado en cuenta. La sentencia *scanf ("%f %*c %d", &x, &y);* lee un real y lo almacena en *x*, lee un carácter y no lo toma en cuenta, y lee un entero y lo toma la variable *y*. Si se introduce *13.89+45* se almacenaría el valor *13.89* en *x* y el valor *45* en *y*.

Es posible limitar el número de caracteres a capturar, escribiendo la cantidad después del símbolo %. Por lo tanto *scanf ("%3s", cadena)*; captura cualquier cadena de caracteres tecleada, pero sólo almacena en

cadena los tres primeros. Sin embargo, el resto de caracteres quedan en el buffer (memoria intermedia) del teclado disponibles.

Ejemplo



```
scanf("%3d %f %*d %2s", &entero, &real, &cadena);
```

Si se captura: 12345 678 5xy4 se almacenarán en las variables los siguientes valores:

entero = 123, real = 45.00, no tomará en cuenta 678 y cadena = "5x"

La siguiente llamada a cualquier función de entrada comenzará a partir de *y*.

A.1.2 Salida de datos: función *printf()*

La *función printf()* es similar a la función *scanf()*, pero se usa para la salida; permite escribir textos y datos en la pantalla con determinado formato.

Sintaxis: *printf ("cadena de control", argumento1,argumento2...);*

La *cadena de control* debe escribirse entre comillas (" ") y puede tener al menos uno de los tres elementos siguientes:

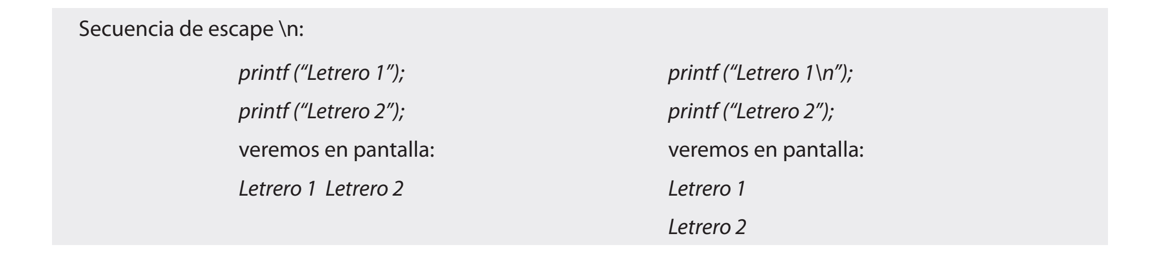
1. Los caracteres que se imprimirán en pantalla (los cuales se visualizarán idénticos).
2. *Secuencias de escape* (vea tabla A.2).
3. Uno o varios *códigos de formato* (indica de qué tipo de dato se trata); dichos códigos se muestran en la tabla A1, los cuales determinan cómo se verán en pantalla los argumentos.

Tabla A.2 Secuencias de escape

Secuencia de escape	***Significado***
\a	Alerta
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Barra invertida
\'	Comilla simple
\"	Comillas dobles
\ooo	Visualiza un carácter cuyo código ASCII es OOO en octal
\xhhh	Visualiza un carácter cuyo código ASCII es HHH en hexadecimal

Ejemplo





Secuencia de escape \n:

printf ("Letrero 1");	*printf ("Letrero 1\n");*
printf ("Letrero 2");	*printf ("Letrero 2");*
veremos en pantalla:	veremos en pantalla:
Letrero 1 Letrero 2	*Letrero 1*
	Letrero 2

Al igual que en la sentencia *scanf* un *especificador o código de formato* se inicia con un carácter % y termina con un carácter de conversión; por ejemplo, en el formato %s la letra ese (*s*) es el carácter de conversión para imprimir cadenas. Debe haber el mismo número de *códigos de formato* que de *argumentos* y cada uno de éstos debe tener un orden. En caso contrario el resultado no será correcto.

Los *argumentos* pueden ser variables, constantes, o en general expresiones de salida. Para imprimir las letras XYZ en la pantalla, podría hacerse de las siguientes formas:

- printf ("XYZ"); //La lista de argumentos es opcional.
- printf ("%s"," XYZ"); //El formato %*s* imprime el texto "XYZ" como *cadena* de caracteres.
- printf("*%c%c%c*" , ' X', ' Y', ' Z'); //Los apóstrofos que encierran cada letra se emplean para designar constantes de caracteres. El formato *%c* imprime el valor como un carácter.

Los *códigos de formato* pueden controlar aspectos de la apariencia de los datos: la longitud, el número de decimales y la justificación a izquierda o derecha.

Sintaxis: % [signo] [longitud] [.precisión] [l/L] Carácter de Conversión

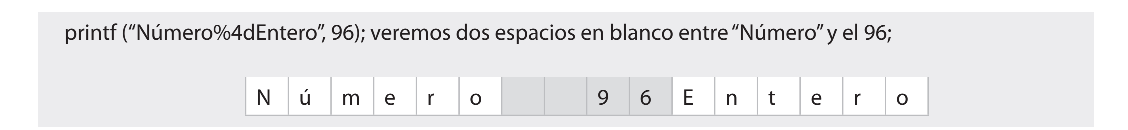
Los valores entre [ ] son opcionales, donde:

signo: indica si el valor aparecerá a la izquierda (signo menos), o a la derecha (por defecto).
longitud: longitud máxima del valor que aparecerá en la pantalla.
precisión: indica el número máximo de decimales que tendrá el valor.
l/L: utilizamos *l* cuando se trata de una variable de tipo *long*, y *L* cuando es de tipo *double*.

El *código de formato %4d* indica que imprimiremos un número entero en cuatro posiciones o lugares, justificando a la derecha. El relleno es blanco.







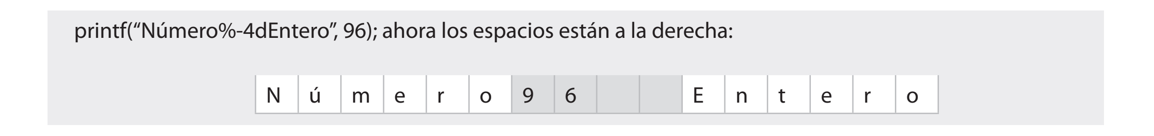
Para rellenar con ceros en lugar de espacios en blanco, colocamos un 0 antes del indicador de tamaño: printf("Número%04dEntero", 96); esto no se puede lograr con justificación izquierda:



Si se desea salida formateada justificada a la izquierda se debe colocar un guión después del signo %.







Si una cadena o número es mayor que el ancho especificado, éstos se imprimirán completos.

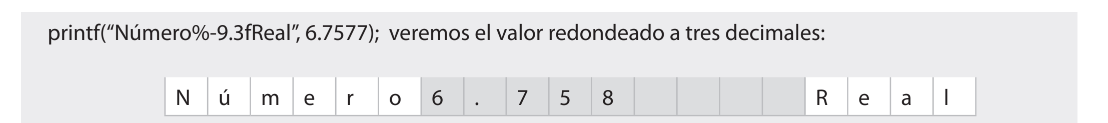
Ejemplo


La *sintaxis* del formato de un *número con parte decimal es:* ***%l.df***
donde *l* es la longitud total del campo (incluyendo el punto decimal) y *d* el número de decimales.

El especificador de formato *%9.3f* nos dice que nuestra variable ocupará nueve lugares: cinco para la parte entera, uno para el punto y tres para los decimales.



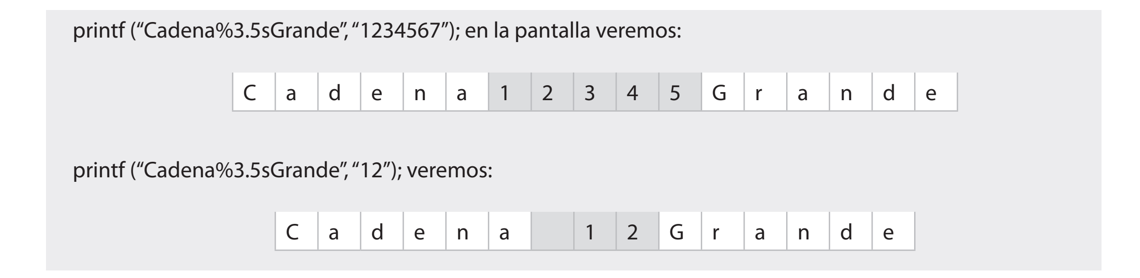




Cuando se usa el punto en una cadena el valor que le sigue al punto indica el máximo ancho de la cadena. El formato *%3.5s* nos dice que la cadena tendrá al menos tres caracteres y un máximo de cinco. Si la cadena sobrepasa el tamaño máximo, se trunca por la derecha.

Ejemplos





printf ("Cadena%3.5sGrande", "1234567"); en la pantalla veremos:

C	a	d	e	n	a	1	2	3	4	5	G	r	a	n	d	e

printf ("Cadena%3.5sGrande", "12"); veremos:

C	a	d	e	n	a		1	2	G	r	a	n	d	e

A.2 Entrada y salida sin formato

Las funciones principales que realizan la entrada y salida sin formato son:

getche() Lee un carácter del teclado, *no* hay que esperar hasta que se pulse la tecla <enter>. Visualiza el eco del carácter.
getchar() Lee un carácter del teclado, espera hasta que se pulsa la tecla <enter>.
getch() Lee un carácter del teclado, *no* hay que esperar que se pulse la tecla <enter>. *No* visualiza el eco del carácter.
putchar() Imprime un carácter en la pantalla en la posición actual del cursor.
gets() Lee una cadena de caracteres introducida por el teclado.
puts() Imprime una cadena en la pantalla, seguida de un carácter de salto de línea.

La función gets() la utilizamos en ejemplos del capítulo 5, para comprender las restantes funciones de entrada veamos el siguiente programa, el cual lee un carácter del teclado y lo imprime en la pantalla:

```
#include  <stdio.h>
#include  <conio.h>
main()
{
   char c;
   puts("Dame un carácter:  ");    fflush(stdin);
   c=getche();  //Lee un carácter del teclado
   putchar(c);  //Lo imprime en la pantalla
   puts("Dame otro carácter:  ");    fflush(stdout);
   c=getchar(); //Lee un carácter del teclado, espera hasta que se pulse la tecla <enter>
   putchar(c);   //Lo imprime en la pantalla
   puts("Dame otro carácter:  ");    fflush(stdin);
   c=getch();    //Lee un carácter del teclado
   putchar(c);    //Lo imprime en la pantalla
   getch(); return  0;
}
```

Una posible salida en pantalla después de ejecutar el programa, sería:

Dame un carácter:

aaDame otro carácter:

b

bDame otro carácter:

c

fflush (stdin)
Limpia el buffer (memoria intermedia) utilizado por la entrada estándar del sistema, generalmente el teclado.

fflush (stdout)
La función *printf* no escribe directamente en la pantalla, sino en una memoria intermedia (buffer). Cuando este buffer (memoria intermedia) se llena o cuando oprimimos un carácter '\n' es cuando se envía el texto a la pantalla. La función *fflush(stdout)* lo que hace es enviar a la pantalla lo que hay en ese buffer.



Diferencias entre Turbo C, Dev-Cpp y Code::Blocks



Por lo general cuando las personas se inician en la programación suelen usar el compilador Borland C, por la ayuda que trae incluida. Uno de los problemas de usar Borland Turbo C o Borland C es que la librería <conio.h> no es portable y sólo se utiliza en las plataformas MS-DOS y Windows (Windows Vista y posteriores no son compatibles con Turbo C 3.0); además usa entorno DOS y no se adecua al sistema gráfico que se suele usar en aplicaciones Windows.

En septiembre de 2006, Borland lanzó una versión recortada del C++Builder para Windows (Turbo C++ for Windows); estaba disponible en dos ediciones: una gratuita, Explorer, y otra de pago, la Pro. Desde octubre de 2009 ya no es posible descargar Turbo C++ Explorer y tampoco se puede obtener la licencia para usar la versión Pro. Sin embargo, existen IDEs libres[1] como Eclipse, Dev C++ y Code::Blocks. A continuación veremos sus diferencias.

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar "Aplicaciones de Cliente Enriquecido". Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit y el compilador que se entrega como parte de Eclipse.

Eclipse sirve para desarrollar programas que estén principalmente orientados a objetivos y para el desarrollo de software[2].

Dev C++

Es un sencillo IDE (Entorno de Desarrollo Integrado) para C y C++, desarrollado en Delphi para Windows y creado por Bloodshed, que se puede descargar solo o con el compilador MinGW (GCC 3.4.2). La última versión es la 4.9.9.2 (febrero de 2005). Permite instalar librerías, usando paquetes .devpak. No podemos diseñar GUI con esta herramienta, pero wxDev C++ incluye un diseñador gráfico de ventanas usando wxWidgets. Ambos programas son software libre, bajo la GNU GPL.

Code::Blocks

Surge en 2005 como una alternativa a Dev-C++. Está escrito en C++, por lo que es multiplataforma y funciona tanto en Windows como Linux. Tiene una arquitectura de Plugins que nos permite diseñar GUI mediante su plugin wxSmith, basado en wxWidgets. La primera versión estable fue la 8.02 (febrero de 2008) y la última la 10.05 (mayo de 2010). Está basado en la plataforma de interfaces gráficas WxWidgets y licenciado bajo la licencia pública general de GNU. Soporta los paquetes de librerías que usa el Dev-Cpp y sin necesidad de descargarlos manualmente. Maneja la sintaxis de C/C++, Fortran, Python, HTML, LaTeX, varios tipos de shell, etcétera.

Las librerías conio.h y graphics.h pertenecen a Borland (no son estándares) y sólo se pueden usar con el compilador de Borland Turbo C o Borland C. Son incompatibles con gcc (Devcpp, Code::Blocks y cualquier IDE de Linux). En Visual C tampoco podemos utilizarlas.

En Dev-C++ y Code::Blocks la emulación de conio del original de Borland® la hacen de forma incompleta; sobre todo en funciones de ventanas, lecturas sin eco en pantalla, y en funciones como “kbhit”.

En Dev-C++ y Code::Blocks en lugar de utilizar la librería graphics.h se emplean alternativas como la API de Windows + GDI, o alguna librería gráfica como SDL, Allegro, inclusive DirectX u OpenGL[3].

En la tabla B.1 presentamos las diferencias más significativas de turbo C, Dev-Cpp y Code::Blocks en los temas vistos en el presente libro.

---

[1] http://brunoprog64.blogsome.com/2007/7/24/ides-en-universidades-peruanas/.

[2] http://freeclipse.foroactivo.com/introduccion-f7/introduccion-lo-basico-de-eclipse-t5.htm.

[3] http://foros.solocodigo.com/viewtopic.php?f=18&t=26779.

Tabla B.1 Diferencias más significativas de Turbo C, Dev-Cpp y Code::Blocks.

TURBO C	*Dev-Cpp Y Code::Blocks*
Plataforma	
MS-DOS y Windows (excepto Windows Vista)	Dev-C++: Windows Code::blocks: Windows y Linux
Tipo de software	
	Libre
Tipo de dato int	
−32,768 a 32,767 (2 bytes)	−2,147,483,648 a 2,147,483,647 (4 bytes)
Tipo de dato booleano	
No existe el equivalente pero se puede implementar con un número entero, 0 es falso y cualquier número diferente de cero verdadero	bool
Programa principal	
void main() Si en el main() se omite el void, es recomendable antes del fin del mismo utilizar return 0, en caso contrario el compilador enviará una advertencia de que cualquier función debe regresar algún valor; aunque se puede seguir trabajando sin ningún problema.	int main() main()
Declaración de arreglos	
El índice debe ser una constante. Vea la página 131.	El índice puede ser una variable. Vea la página 132.
Uso de system (librería stdlib.h) sin utilizar conio2 en Dev C++ y Code::Blocks	
clrscr() getch() o getche()	system("CLS") system("PAUSE")
textcolor() textbackground()	system("color #C"); donde: # es el color de fondo (0 a 9) y C es el color de la letra (A a la F, minúscula o mayúscula).
Librería conio limitada en Dev-Cpp y Code::Blocks	
conio clrscr() gotoxy() textcolor() textbackground()	*conio2* clrscr() gotoxy() textcolor() textbackground()
Parpadeo de texto	
BLINK	No existe equivalente
Funciones aleatorias	
randomize() y random()	srand() y rand()
Detener la ejecución del programa una cantidad de milisegundos	
delay(100)	Sleep(100)
Declaración de un arreglo	
Tamaño constante int a[10]; float ×[10][10];	*Tamaño constante y variable* int a[10]; float ×[10][10]; scanf("%d%d",&n,&m); int a[n]; float ×[n][m];



Anexo C

Aplicaciones

En el presente anexo, presentamos aplicaciones en diferentes áreas de Ciencias Exactas e Ingenierías; dos de ellas son parte del proyecto final presentado por alumnos en el curso de Programación Estructurada.

1. **Química.** Con ayuda de un menú crear los datos de los diferentes elementos de la tabla periódica. Dicho menú contendrá la información siguiente: 1) Capturar, 2) Mostrar, 3) Buscar, 4) Eliminar, 5) Modificar, 6) Salir.

```
#include <conio2.h>
#include <stdio.h>
struct elemento
{
   char Nombre[15],Valencia[10],Edo_Ox[2],ElecNeg[3], Conf_Elec[25],eV[5];
   char M_Atom[7],Den[5],Pto_Ebu[6],Pto_Fus[6];
   int No_atomico;
}e[118];
int cont;
int buscar(int x);
void capturar()
{
   printf ("Nombre:  ");fflush(stdin);gets(e[cont].Nombre);
   printf ("Número atómico:  ");scanf("%d",&e[cont].No_atomico);
   printf ("Valencia:  ");fflush(stdin);gets(e[cont].Valencia);
   printf ("Estado de oxidación:  ");fflush(stdin);gets(e[cont].Edo_Ox);
   printf ("Electronegatividad:  ");fflush(stdin);gets(e[cont].ElecNeg);
   printf ("Configuración electrónica:  ");fflush(stdin);gets(e[cont].Conf_Elec);
   printf ("Primer potencial de ionización (eV): ");fflush(stdin);gets(e[cont].eV);
   printf ("Masa atómica  (g/mol):  ");fflush(stdin);gets(e[cont].M_Atom);
   printf ("Densidad  (g/ml):  ");fflush(stdin);gets(e[cont].Den);
   cont=cont+1;
}
void mostrar()
{
   for  (int i=0;i<cont;i++)
   {
        printf ("\nNombre:  %s",e[i].Nombre);
        printf ("\nNúmero atómico:  %d",e[i].No_atomico);
        printf ("\nValencia:  %s",e[i].Valencia);
        printf ("\nEstado de oxidación:  %s",e[i].Edo_Ox);
        printf ("\nElectronegatividad:  %s",e[i].ElecNeg);
        printf ("\nConfiguración electrónica:  %s",e[i].Conf_Elec);
        printf ("\nPrimer potencial de ionización  (eV):  %s  ",e[i].eV);
        printf ("\nMasa atómica  (g/mol):  %s",e[i].M_Atom);
        printf ("\nDensidad  (g/ml):  %s",e[i].Den);
        printf ("\nOPRIME ENTER PARA CONTINUAR");getch();clrscr();
   }
}
void modificar()
```

(*continúa*)

(*continuación*)

```
{
    int op,i,p;
    p=buscar(5);
    if (p!=-1)
    {
    do
    {
    printf ("\n1)Nombre:  %s",e[p].Nombre);
    printf ("\n2)Número atómico:  %d",e[p].No_atomico);
    printf ("\n3)Valencia:  %s",e[p].Valencia);
    printf ("\n4)Estado de oxidación:  %s",e[p].Edo_Ox);
    printf ("\n5)Electronegatividad:  %s",e[p].ElecNeg);
    printf ("\n6)Configuración electrónica:  %s",e[p].Conf_Elec);
    printf ("\n7)Primer potencial de ionización  (eV):  %s  ",e[p].eV);
    printf ("\n8)Masa atómica  (g/mol):  %s",e[p].M_Atom);
    printf ("\n9)Densidad  (g/ml):  %s",e[p].Den);
    printf ("\n10)SALIR)  ");
    printf ("\n¿Qué Deseas Modificar?");
    scanf("%d",&op);
    switch(op)
    {
    case 1: printf ("\nNombre:  ");fflush(stdin);gets(e[p].Nombre); break;
    case 2: printf ("\nNúmero atómico:  ");scanf("%d",&e[p].No_atomico); break;
    case 3: printf ("\nValencia:  ");fflush(stdin);gets(e[p].Valencia); break;
    case 4: printf ("\nEstado de oxidación:  ");fflush(stdin);gets(e[p].Edo_Ox); break;
    case 5: printf ("\nElectronegatividad:  ");fflush(stdin);gets(e[p].ElecNeg); break;
    case 6: printf ("\nConfiguración electrónica: ");fflush(stdin);gets(e[p].Conf_Elec); break;
    case 7: printf ("\nPrimer potencial de ionización (eV): "); fflush(stdin); gets (e[p].eV);
      break;
    case 8:printf ("\nMasa atómica  (g/mol):  ");fflush(stdin);gets(e[p].M_Atom); break;
    case 9:printf ("\nDensidad  (g/ml):  ");fflush(stdin);gets(e[p].Den); break;
    }
    }while (op!=10);
    }
}
void eliminar()
{
    int p,i;
    char r;
    p=buscar(4);
    if (p!=-1)
    {
    printf("\n¿Lo Deseas Eliminar (s/n)?  ");
    scanf("%s",&r);
    if(r=='s'||r=='S')
    {
```

(*continúa*)

(*continuación*)

```
    for (i=p;i<=cont;i++)
    e[i]= e[i+1];
    cont= cont-1;
    }
    }
}
int buscar(int x)
{
    int NA,b=0,pos;
    if (x==4) printf("\n Dame el NUMERO ATOMICO del elemento a ELIMINAR:");
    else if (x==5) printf("\n Dame el NUMERO ATOMICO del elemento a MODIFICAR:");
    else printf("\n Dame el NUMERO ATOMICO del elemento a BUSCAR:");
    scanf("%d",&NA); if (cont==0)
          printf("\n NO EXISTE EL REGISTRO\n");
          else
          for (int i=0;i<cont;i++)
          if(NA==e[i].No_atomico)
          {if (x!=5)
          {
                printf ("\nNombre:  %s",e[i].Nombre);
                printf ("\nNúmero atómico:  %d",e[i].No_atomico);
                printf ("\nValencia:  %s",e[i].Valencia);
                printf ("\nEstado de oxidación:  %s",e[i].Edo_Ox);
                printf ("\nElectronegatividad:  %s",e[i].ElecNeg);
                printf ("\nConfiguración electrónica:  %s",e[i].Conf_Elec);
                printf ("\nPrimer potencial de ionización  (eV):  %s  ",e[i].eV);
                printf ("\nMasa atómica  (g/mol):  %s",e[i].M_Atom);
                printf ("\nDensidad  (g/ml):  %s",e[i].Den);
          }
          b=1;pos=i;break;
          }
    if (b==0) {printf ("NO EXISTE REGISTRO");pos=-1;}
    if (x==3) {printf ("\nOPRIME ENTER PARA CONTINUAR");getch();}
    return (pos);
}
main()
{
    int opc; cont=0;
    do
    {
    clrscr();
    printf("\n 1)CAPTURAR \n 2)MOSTRAR\n 3)BUSCAR\n 4)ELIMINAR\n 5)MODIFICAR\n 6)
      SALIR\n"); scanf("%d",&opc);
    if (opc==1) capturar();
```

(*continúa*)

(*continuación*)

```
    if (opc==2) mostrar();
    if (opc==3) buscar(3);
    if (opc==4) eliminar();
    if (opc==5) modificar();
    }while (opc!=6);
}
```

2. **Ingeniería en Comunicaciones y Electrónica e Ingeniería Mecánica Eléctrica.** Con ayuda de un menú calcular la Ley de Ohm y la dilatación lineal del fierro en un rango de temperatura dado. Dicho menú contendrá la información siguiente: 1) Intensidad, voltaje o resistencia de una corriente eléctrica, 2) Dilatación lineal del fierro en un rango de temperatura dado, 3) Salir.

```
/* Leon Flores Roberto  */
#include <stdio.h>
#include <conio2.h>
#define cdl 0.000012
void leyohm (int x);
float term (float x, float y);
main ()
{
    clrscr();
    int opc,n;
    float tf,ti,mi,t1;
    do
    {
        clrscr();
        printf("1)Intensidad, voltaje o resistencia de una corriente eléctrica\n");
        printf("2)Dilatación lineal del fierro en un rango de temperatura dado\n3)Salir\n");
        printf("Elige tu opción:  ");    scanf("%d",&opc);
        if(opc==1)
        {
             clrscr();
             printf("Qué desea calcular:\n\n1)Intensidad\n\n2)Voltaje\n\n3)Resistencia\n\n");
             printf("Elige tu opción:  ");  scanf("%d",&n); leyohm(n);
        }
             else if(opc==2)
                {
                    clrscr(); printf("Dilatación lineal del fierro\n");
                    printf("Dame la temperatura inicial en  °C:  "); scanf("%f",&ti);
                    printf("Dame la temperatura final en  °C:  "); scanf("%f",&tf);
                    printf("Dame la medida inicial:  "); scanf("%f",&mi);
                    t1=ti;
                    while(t1!=tf)
                    {
                        printf("En  %.2f °C el fierro se dilató de %.2f hasta ",t1,mi);
```

(*continúa*)

(*continuación*)

```
                    printf("%.4f\n",term(mi,t1));
                    if((tf-ti)>0) t1+=1;
                        else if((tf-ti)<0) t1-=1;
                  } getch();
              }
    }
    while(opc!=3);
}
void leyohm(int x)
{
    clrscr();    float vol,res,in;
    if(x==1)
        {
            printf("Intensidad\n");
            printf("Dame el voltaje y la resistencia:  \n");
            scanf("%f  %f", &vol, &res); in=vol/res;
            printf("\nLa intensidad de la corriente eléctrica es:  %.2f amperes",in);
        }
    if(x==2)
        {
            printf("Voltaje\n");
            printf("Dame la intensidad y la resistencia:  \n");
            scanf("%f  %f", &in, &res); vol=in*res;
            printf("\nEl valor del voltaje de la corriente eléctrica es:  %.2f volts",vol);
        }
    if(x==3)
        {
            printf("Resistencia\n");
            printf("Deme el valor del voltaje y de la intensidad:  \n");
            scanf("%f  %f",&vol,&in);
            res=vol/in;
            printf("\nEl valor de la resistencia de la corriente eléctrica es:  %.2f
                    ohms",res);
        } getch();
}
float term(float x, float y)
{
    float mf;
    mf=(x*cdl*y)+x;
    return mf;
}
```

3. **Ingeniería Civil.** Con ayuda de un menú calcular la flecha máxima de tres tipos de vigas de acero (IPS) simplemente apoyadas y con carga concentrada. La **viga** es el elemento estructural utilizado para cubrir espacios (techos) y soportar el peso colocado encima del elemento. El predimensionado de las vigas consiste en determinar las dimensiones necesarias para que el elemento sea capaz de resistir la flexión

y el corte, así como también debe tener dimensiones tales que la **flecha** (desplazamiento vertical) no sea excesiva. Para el cálculo de la flecha se emplea el módulo de elasticidad del material utilizado (por ejemplo acero) y el momento de inercia del perfil de la viga, la **flecha máxima** *(f)* se compara con los valores admisibles (manual del fabricante) para ese tipo de estructuras[1]. En este caso en particular nos referimos sólo a vigas simplemente apoyadas con carga concentrada *(F)*; para poder calcular la flecha máxima debemos analizar la siguiente figura C.1 y tabla C.1.

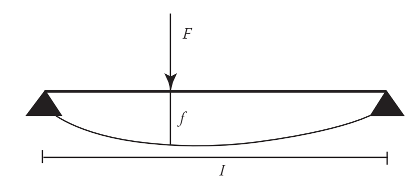
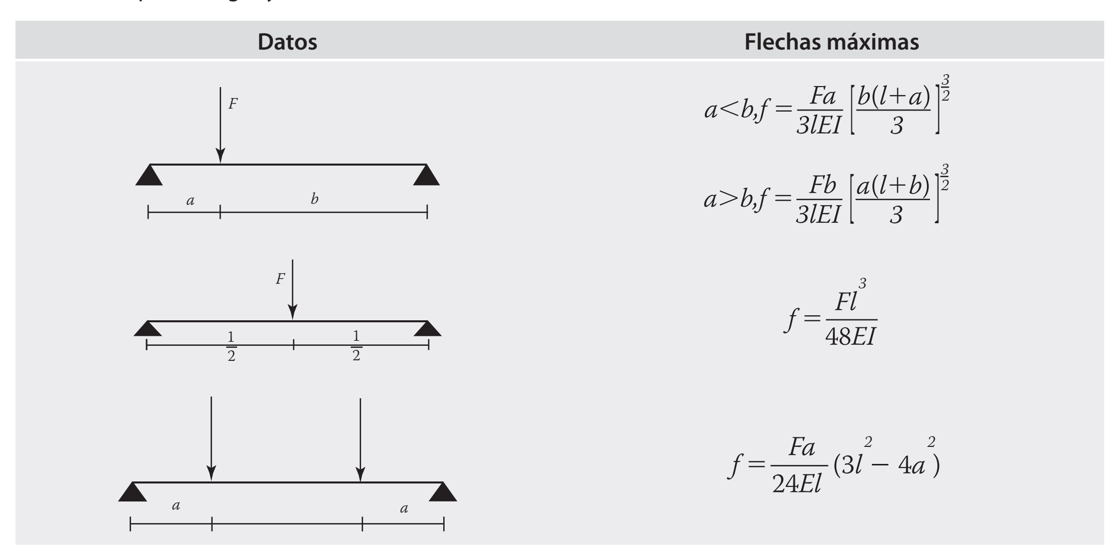


Figura C.1 Viga simplemente apoyada con carga concentrada.

Tabla C.1 Tipos de vigas y sus flechas máximas



Datos	Flechas máximas
F; a; b	$a<b, f=\frac{Fa}{3lEI}\left[\frac{b(l+a)}{3}\right]^{\frac{3}{2}}$ $a>b, f=\frac{Fb}{3lEI}\left[\frac{a(l+b)}{3}\right]^{\frac{3}{2}}$
F; $\frac{1}{2}$; $\frac{1}{2}$	$f=\frac{Fl^3}{48EI}$
a; a	$f=\frac{Fa}{24El}(3l^2-4a^2)$

Fuente: http://citywiki.ugr.es/wiki/Imagen:Formulario_Vigas.pdf

Donde:

F = Fuerza aplicada
a = Distancia de la fuerza al apoyo izquierdo
b = Distancia de la fuerza al apoyo derecho
f = Flecha máxima de la viga
E = Módulo de elasticidad del material
I = Momento de inercia del perfil de la viga

Un ejemplo de viga de acero muy utilizada en la construcción de casas habitación lo vemos en la figura C2.

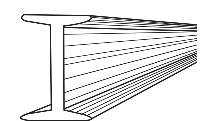


Figura C.2 Viga de acero tipo IPS.

[1] Segui, W. (2000). *Diseño de estructuras de acero con LRFD.* México D.F., México: Internacional Thomson Editores, S.A. de C.V.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define E  2100000  //Módulo de elasticidad del acero  (Kg/cm2)
#define I  249.7  //Viga de  4”
void Viga1(float x,float y,float z, float w);//Prototipo
float Viga3  (float x, float y, float z);//Prototipo
float Viga2  (float x,float y);//Prototipo
main()
{
   int op;
   float a, b, l, F, f;
   do
   {
   printf (“1)Viga  1\n2)Viga  2\n3)Viga  3\n4)Salir\n”);
   printf (“Dame tu opción:  “); scanf  (“%i”, & op);
   if (op>=1 && op<=3) { printf(“\nDame l:  “); scanf (“%f”, &l);  }
   switch (op)
   {
       case 1: printf(“Dame a(en cm.):  “);   scanf  (“%f”, &a);
               printf(“Dame b(en cm.):  “);   scanf  (“%f”, &b);
               printf(“Dame F(en Kg.):  “);   scanf  (“%f”, &F);
               if (a< b) Viga1 (a, b, l, F);
                   else Viga1 (b, a, l, F);
               break;
       case 2: printf(“Dame F(en Kg.):  “);  scanf  (“%f”, &F);
               f=Viga2(l,F); printf  (“Flecha = %f\n”,f);
               break;
       case 3: printf(“Dame a(en cm.):  “);    scanf  (“%f”, &a);
               printf(“Dame F(en Kg.):  “);    scanf  (“%f”, &F);
               f=Viga3(a, l, F); printf  (“Flecha =  %f\n”,f);
               break;
   }
   }
   while  (op!= 4);
}
void Viga1  (float x, float y, float z, float w)
{
   float R;
   R=  (w*x/(3*z*E*I))*pow(y*(z+x)/3,1.5);
   printf  (“Flecha = %f\n”,R);
}
float Viga3  (float x, float y, float z)
{
   float R;
```

(continúa)

(*continuación*)

```
    R= (z*x/(24*E*I))*(3*y*y−4*x*x);
    return (R);
}
float Viga2  (float x, float y)
{
    float R;
    R= y*x*x*x/(48*E*I);
    return (R);
}
```

4. **Ingeniería Civil y Topografía**. Con ayuda de un menú calcular la altura de un edificio y el levantamiento topográfico por método de radiaciones. Dicho menú contendrá la información siguiente: 1) Nivelación trigonométrica: Altura de un edificio, 2) Levantamiento topográfico por el método de radiaciones.

    La altura de un edificio consiste en la aplicación de los principios básicos de la altimetría topográfica con el objetivo de determinar la altura de una construcción utilizando identidades trigonométricas, leyes de los senos y teorema de Pitágoras.

    El levantamiento topográfico por el método de radiaciones es el método de levantamiento más utilizado en la Topografía e Ingeniería Civil. Consiste en la obtención de las coordenadas rectangulares de un terreno sobre un plano de coordenadas cartesianas, la superficie, perímetro y rumbos hasta los vértices del deslinde a partir de los azimuts y las distancias desde un punto en el interior del terreno hasta cada uno de los vértices de su contorno.

```
/*Rosas Zepeda Alberto Francisco y Velasco Cardona Eder Omar*/
#include<stdio.h>
#include<conio2.h>
#include<math.h>
#define PI  3.1416
void levantamiento_radiaciones();
void nivelacion_trigonometrica();
float conversion_angulo();
void azimuts(float a[20],int n);
void distancias(float d[20],int n);
void transformacion(float a[20],float rbo[20],int n);
void imprimir_datos(float a[20], float d[20],float rbo[20],int n);
void coordenadas(float d[20], float a[20], int n, float norte[20], float este[20]);
void superficie(float norte[20], float este[20], int n);
void dist_vertices(int n, float norte[20], float este[20]);
main()
{
    textbackground(RED); textcolor(15);
    clrscr(); int opcion;
    do
    {
       clrscr();
       /*OPCIONES DE APLICACION*/
```

(*continúa*)

(*continuación*)

```
      gotoxy(16,4);  printf("APLICACIONES EN INGENIERIA CIVIL Y TOPOGRAFIA\n");
      gotoxy(14,8);  printf("1) NIVELACION TRIGONOMETRICA: ALTURA DE UN EDIFICIO\n");
      gotoxy(14,10);
      printf("2) LEVANTAMIENTO TOPOGRAFICO POR EL METODO DE RADIACIONES\n");
      gotoxy(14,12); printf("3) SALIR");
      gotoxy(31,19); printf("Elige una opción:  ");
      gotoxy(49,19);scanf("%d",&opcion);clrscr();
      switch(opcion)
      {  case  1: nivelacion_trigonometrica();
         break;
         case  2: levantamiento_radiaciones(); getch(); break;
      }
   }
   while(opcion!=3);
}
void levantamiento_radiaciones()
{
   int n;    float ang[20],dist[20],rumbo[20], norte[20], este[20];
   printf("\n\n\tLEVANTAMIENTO TOPOGRAFICO POR EL METODO DE RADIACIONES");
   printf("\n\n\tIntroduzca el número de lados del terreno:   ");  scanf("%d",&n);
   printf("\n\tSea estación O la posición del teodolito.\n\n");
   azimuts(ang,n); distancias(dist,n);
   transformacion(ang,rumbo,n);    getch();
   imprimir_datos(ang,dist,rumbo,n);    getch();
   coordenadas(dist,ang,n,norte,este);
   superficie(norte,este,n);    getch();
   dist_vertices(n,norte,este);
   printf("\n\nCALCULO FINALIZADO");
   printf("\n\nPresione enter para continuar");
}

/*NIVELACION TRIGONOMETRICA:APLICACION DE TOPOGRAFIA PARA DETERMINAR LA ALTURA DE UN
  EDIFICIO*/
void nivelacion_trigonometrica()
{
   float distAB,dist_horAC,dist_verAC,ang_horA,ang_verA, ang_horB, ang_horC,
     teodolito, altura;
   gotoxy(18,3); printf("NIVELACION TRIGONOMETRICA: ALTURA DE UN EDIFICIO");
   gotoxy(8,5); printf("Al introducir los ángulos horizontales, el ángulo vertical");
   printf(" y distancia requeridos podrás obtener la altura del edificio en
     cuestión");
   gotoxy(3,8); printf("\tSean:\n\tA:Estación principal\n\tB:Estación auxiliar\n");
   printf("\tC:Punto más alto del edificio");
   gotoxy(5,12); printf("\n\tIntroduzca la distancia de entre las estaciones A y B:  ");
   scanf("%f",&distAB);
```

(*continúa*)

(*continuación*)

```
    printf("\n\tIntroduzca el ángulo horizontal A  (de la estación B a C):  ");
    ang_horA  = conversion_angulo();
    printf("\n\tIntroduzca el ángulo vertical A  (de la estación A a C):  ");
    ang_verA  = conversion_angulo();
    printf("\n\tIntroduzca el ángulo horizontal B  (de la estación A a C):  ");
    ang_horB  = conversion_angulo();
/*Entre A, B y C se forma un triángulo, la suma de sus angulos internos es  180°, por
  lo que:  */
    ang_horC=(180-(ang_horA+ang_horB));
    printf("\n\n\tEl angulo horizontal C es:  %.4f grados",ang_horC);
/*Ahora determinamos la distancia horizontal desde A hasta C utilizando leyes de los
  senos*/
    dist_horAC=(sin(ang_horA*PI/180)*distAB)/sin(ang_horC*PI/180);
    printf("\n\n\tLa distancia horizontal A-C es:  %.4f m",dist_horAC);
/*Con la distancia A-C horizontal podemos obtener la distancia A-C vertical
  utilizando el teorema de pitágoras  */
    dist_verAC=tan(ang_verA*PI/180)*dist_horAC;
    printf("\n\n\tLa altura desde el lente del aparato hasta C es:  %.4f m",dist_verAC);
/*Para obtener la altura total, sumamos altura AC+altura del aparato*/
    printf("\n\n\tIntroduce la altura del aparato: "); scanf("%f",&teodolito);
    altura = dist_verAC+teodolito; printf("\n\n\t\t\tALTURA DEL EDIFICIO = %.4f m", altura);
    printf("\n\nCALCULO FINALIZADO");
    printf("\n\nPresione enter para continuar");
    getch();
}
/*ANGULO:CAPTURA UN ANGULO EN GRADOS Y MINUTOS Y LAS OPERACIONES CORRESPONDIENTES PARA
  TRANSFORMARLO A UN SOLO NUMERO REAL*/
float conversion_angulo()
{
    float grados, minutos, decimal, angulo;
    printf("\n\tGrados:    "); scanf("%f",&grados);
    printf("\tMinutos:    "); scanf("%f",&minutos);
    decimal=minutos/60;
    angulo=grados+decimal;
    return angulo;
}
/*AZIMUTS:CAPTURA LOS ANGULOS DE AZIMUT DESDE LA ESTACION CENTRAL HASTA CADA VERTICE
  DEL TERRENO*/
void azimuts(float a[20],int n)
{
   for  (int i=0;i<n;i++)
   {
     printf("\n\tIntroduzca el azimut de la estación  0 a la estación  %d:  ",i+1);
       a[i]=conversion_angulo();
   }
}
```

(*continúa*)

(*continuación*)

```
/*DISTANCIAS:CAPTURA LAS DISTANCIAS DESDE LA ESTACION CENTRAL HASTA CADA VERTICE DEL
  TERRENO*/
void distancias(float d[20],int n)
{
   printf("\n");
   for  (int i=0;i<n;i++)
   {
      printf("\tIntroduzca la distancia de la estación  0 a la estación %d: ",i+1);
         scanf("%f", &d[i]);
   }
}
/*IMPRIMIR_DATOS: IMPRIME LOS DATOS DE ENTRADA A MANERA DE TABLA*/
void imprimir_datos(float a[20], float d[20], float rbo[20], int n)
{
   printf("\n\n  Estación  Punto Visado  Azimut  Distancia  Rumbo\n");
   for(int i=0;i<n;i++)
   printf("O\t\t%d\t\t%.3f\t\t%.3f\t\t%.3f\n",i+1,a[i],d[i],rbo[i]);
}
/*TRANSFORMACION: CAMBIA LOS GRADOS AZIMUT DE O A CADA VERTICE DEL TERRENO A GRADOS
  RUMBO*/ void transformacion(float a[20], float rbo[20], int n)
{
   printf("\nConversión de los azimuts desde O a cada vértice, a rumbos:\n");
   for(int i=0;i<n;i++)
   {
        if(a[i]>0&&a[i]<90)
        {
           rbo[i]=a[i];
           printf("%.3f azimut  = rumbo  %.3f grados Noreste", a[i], rbo[i]);
        }
        if(a[i]>90&&a[i]<180)
        {
           rbo[i]=180-a[i];
           printf("%.3f azimut = rumbo  %.3f grados Sureste",a[i],rbo[i]);
        }
        if(a[i]>180&&a[i]<270)
        {
           rbo[i]=a[i]-180;
           printf("%.3f azimut  = rumbo  %.3f grados Suroeste",a[i],rbo[i]);
        }
        if(a[i]>270&&a[i]<360)
        {
           rbo[i]=360-a[i];
           printf("%.3f azimut = rumbo  %.3f grados Noroeste",a[i],rbo[i]);
        }
        printf("\n");
   }
}
```

(*continúa*)

(*continuación*)

```
/*COORDENADAS:TRANSFORMA LAS POSICIONES DE LOS VERTICES DEL TERRENO DEFINIDAS MEDIANTE
  AZIMUTS O RUMBOS Y DISTANCIAS, A POSICIONES POR COORDENADAS RECTANGULARES  */
void coordenadas(float d[20], float a[20], int n, float nor[20], float este[20])
{
   printf("\n     Tomando en cuenta al punto O como origen en la posición  (0,0)");
   for(int i=0;i<n;i++)
   {
        nor[i]=cos(a[i]*PI/180)*d[i];
        este[i]=sin(a[i]*PI/180)*d[i];
        printf("\n      Coordenadas rectangulares del punto  %d\t",i+1);
        printf(" nor:  %.3f\tEste:  %.3f",nor[i],este[i]);
   }
}
/*SUPERFICIE: CALCULA EL AREA DEL TERRENO POR PRODUCTO CRUZADO DE COORDENADAS
  RECTANGULARES*/
void superficie(float nor[20], float este[20], int n)
{
   float p1[20],p2[20],acump1=0,acump2=0,totalp1,totalp2,area;
   for(int i=0;i<n-1;i++)
   {
        p1[i]=nor[i]*este[i+1];
        p2[i]=nor[i+1]*este[i];
        acump1=acump1+p1[i];
        acump2=acump2+p2[i];
   }
   totalp1=acump1+nor[n-1]*este[0];
   totalp2=acump2+nor[0]*este[n-1];
   if(totalp1>totalp2) area=(totalp1-totalp2)/2;
        else area=(totalp2-totalp1)/2;
   printf("\n\nLA SUPERFICIE DEL TERRENO ES:  %f m2",area);
}
/*DIST_VER: CALCULA LA DISTANCIA ENTRE  2 VERTICES CONTIGUOS Y APROVECHANDO LA
  OBTENCION DE ESTAS DISTANCIAS, CALCULA EL PERIMETRO DE LA FIGURA*/
void dist_vertices(int n, float nor[20], float este[20])
{
   float dist[20], perimetro=0;
   printf("\n\nDistancias entre los vértices del terreno:");
   for(int i=0;i<n-1;i++)
   {
   dist[i]=sqrt((nor[i+1]-nor[i])*(nor[i+1]-nor[i])+(este[i+1]-este[i])*(este[i+1]
     -este[i]));
   printf("\nDistancia  %d  -  %d  =  %f",i+1,i+2,dist[i]);
   perimetro=perimetro+dist[i];
   }
```

(*continúa*)

(*continuación*)

```
    dist[n−1]=sqrt((nor[0]−nor[n−1])*(nor[0]−nor[n−1])+(este[0]−este[n−1])*
      (este[0]− este[n−1])); printf("\nDistancia  %d − %d  =  %.3f",n,1,dist[n−1]);
    perimetro=perimetro+dist[n−1];    printf("\n\nPERIMETRO  =  %.3f m",perimetro);
}
```



Anexo D

Código ASCII

Dec	Car	Hex	Oct	Dec	Car	Hex	Oct	Dec	Car	Hex	Oct	De c	Car	Hex	Oct	Dec	Car	Hex	Oct	Dec	Car	Hex	Oct
1	☺	1	001	43	+	28	053	85	U	55	125	127	⌂	7F	177	169	®	A9	251	211	Ë	D3	323
2	☻	2	002	44	,	23	054	86	V	56	126	128	Ç	80	200	170	¬	AA	252	212	È	D4	324
3	♥	3	003	45	-	2D	055	87	W	57	127	129	ü	81	201	171	½	AB	253	213	ı	D5	325
4	♦	4	004	46	.	2E	056	88	X	58	130	130	é	82	202	172	¼	AC	254	214	Í	D6	326
5	♣	5	005	47	/	2F	057	89	Y	59	131	131	â	83	203	173	¡	AD	255	215	Î	D7	327
6	♠	6	006	48	0	30	060	90	Z	5A	132	132	ä	84	204	174	«	AE	256	216	Ï	D8	330
7	•	7	007	49	1	31	061	91	[	5B	133	133	à	85	205	175	″	AF	257	217	┘	D9	331
8	◘	8	010	50	2	32	062	92	\	5C	134	134	å	86	206	176	░	B0	260	218	┌	DA	332
9	○	9	011	51	3	33	063	93	]	5D	135	135	ç	87	207	177	▒	B1	261	219	█	DB	333
10	◙	A	012	52	4	34	064	94	^	5E	136	136	ê	88	210	178	▓	B2	262	220	▄	DC	334
11	♂	B	013	53	5	35	065	95	_	5F	137	137	ë	89	211	179	│	B3	263	221	¦	DD	335
12	♀	C	014	54	6	36	066	96	`	60	140	138	è	8A	212	180	┤	B4	264	222	Ì	DE	336
13	♪	D	015	55	7	37	067	97	a	61	141	139	ï	8B	213	181	Á	B5	265	223	▀	DF	337
14	♫	E	016	56	8	38	070	98	b	62	142	140	î	8C	214	182	Â	B6	266	224	Ó	E0	340
15	☼	F	017	57	9	39	071	99	c	63	143	141	ì	8D	215	183	À	B7	267	225	ß	E1	341
16	►	10	020	58	:	3A	072	100	d	64	144	142	Ä	8E	216	184	©	B8	270	226	Ô	E2	342
17	◄	11	021	59	;	3B	073	101	e	65	145	143	Å	8F	217	185	╣	B9	271	227	Ò	E3	343
18	↕	12	022	60	<	3C	074	102	f	66	146	144	É	90	220	186	║	BA	272	228	õ	E4	344
19	‼	13	023	61	=	3D	075	103	g	67	147	145	æ	91	221	187	╗	BB	273	229	Õ	E5	345
20	¶	14	024	62	>	3E	076	104	h	68	150	146	Æ	92	222	188	╝	BC	274	230	µ	E6	346
21	§	15	025	63	?	3F	077	105	i	69	151	147	ô	93	223	189	¢	BD	275	231	þ	E7	347
22	▬	16	026	64	@	40	100	106	j	6A	152	148	ö	94	224	190	¥	BE	276	232	Þ	E8	350
23	↨	17	027	65	A	41	101	107	k	6B	153	149	ò	95	225	191	┐	BF	277	233	Ú	E9	351
24	↑	18	030	66	B	42	102	108	l	6C	154	150	û	96	226	192	└	C0	300	234	Ü	EA	352
25	↓	19	031	67	C	43	103	109	m	6D	155	151	ù	97	227	193	┴	C1	301	235	Û	EB	353
26	→	1A	032	68	D	44	104	110	n	6E	156	152	ÿ	98	230	194	┬	C2	302	236	ý	EC	354
27	←	1B	033	69	E	45	105	111	o	6F	157	153	Ö	99	231	195	├	C3	303	237	Ý	ED	355
28	∟	1C	034	70	F	46	106	112	p	70	160	154	Ü	9A	232	196	─	C4	304	238	¨	EE	356
29	↔	1D	035	71	G	47	107	113	q	71	161	155	ø	9B	233	197	┼	C5	305	239	´	EF	357
30	▲	1E	036	72	H	48	110	114	r	72	162	156	£	9C	234	198	ä	C6	306	240	·	F0	360
31	▼	1F	037	73	I	49	111	115	s	73	163	157	Ø	9D	235	199	Ä	C7	307	241	±	F1	361
32		20	040	74	J	4A	112	116	t	74	164	158	×	9E	236	200	╚	C8	310	242	‗	F2	362
33	¡	21	041	75	K	4B	113	117	u	75	165	159	ƒ	9F	237	201	╔	C9	311	243	¾	F3	363
34	"	22	042	76	L	4C	114	118	v	76	166	160	á	A0	240	202	╩	CA	312	244	¶	F4	364
35	#	23	043	77	M	4D	115	119	w	77	167	161	í	A1	241	203	╦	CB	313	245	§	F5	365
36	$	24	044	78	N	4E	116	120	x	78	170	162	ó	A2	242	204	╠	CC	314	246	÷	F6	366
37	%	25	045	79	O	4F	117	121	y	79	171	163	ú	A3	243	205	═	CD	315	247	„	F7	367
38	&	26	046	80	P	50	120	122	z	7A	172	164	ñ	A4	244	206	╬	CE	316	248	°	F8	370
39	'	27	047	81	Q	51	121	123	{	7B	173	165	Ñ	A5	245	207	¤	CF	317	249	″	F9	371
40	(	28	050	82	R	52	122	124	|	7C	174	166	ª	A6	246	208	ỏ	D0	320	250	.	FA	372
41	)	29	051	83	S	53	123	125	}	7D	175	167	º	A7	247	209	Đ	D1	321	251	¹	FB	373
42	*	2A	052	84	T	54	124	126	~	7E	176	168	¿	A8	250	210	Ê	D2	322	252	³	FC	374
																				253	²	FD	375
																				254	■	FE	376
																				255		FF	377

Los caracteres ASCII no imprimibles se definen mediante la barra invertida (\).

Por ejemplo \", (Ver tabla 2 del Anexo A).



Anexo E

Glosario

Acumulador. Es una variable que acumula sobre sí misma un conjunto de valores, para de esta manera tener la acumulación de todos ellos en una sola variable. Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante.

Algoritmo. Conjunto de pasos ordenados y finitos que permiten resolver un problema o tarea específica. Los algoritmos son independientes del lenguaje de programación y de la computadora que se vaya a emplear para ejecutarlo.

Anidamiento. Consiste en insertar (anidar) una estructura dentro de otra.

Arreglo. Es un tipo de datos estructurado que almacena en una sola variable un conjunto limitado de datos o elementos del mismo tipo. Es un conjunto de localidades de memoria contiguas donde la dirección más baja corresponde al primer elemento y la dirección más alta al último. Por sí mismo, el nombre del arreglo apunta a la dirección del primer elemento del arreglo.

Arreglo bidimensional. Es un conjunto de *n* elementos del mismo tipo almacenados en memoria continua en una matriz o tabla. A diferencia de los arreglos unidimensionales que solo requieren de un subíndice, los arreglos bidimensionales para acceder a cada elemento del arreglo requieren de dos índices o subíndices declarados en dos pares de corchetes, donde el primer corchete se refiere al tamaño de filas y el segundo al tamaño de columnas.

Arreglo unidimensional. Es un conjunto de *n* elementos del mismo tipo almacenados en memoria contigua en un vector o lista; para acceder a cada elemento del arreglo se requiere de un solo índice o subíndice, el cual representa la posición en la que se encuentra.

Bandera, flag, interruptor o marca. Variable muy utilizada en búsquedas que puede tomar sólo dos valores opuestos, generalmente: 1 (verdadero) o 0 (falso), a lo largo de la ejecución del algoritmo o programa.

Bucles, ciclo o iteración. Es un segmento de un algoritmo o programa, cuya(s) instrucción(es) se repite(n) un número conocido o indefinido de veces mientras se cumpla una determinada condición.

Búsqueda binaria. Se basa en el algoritmo divide y vencerás. Para poder utilizar la búsqueda binaria el vector debe estar ordenado. Primeramente se divide el arreglo a la mitad y se compara el elemento central con el número a buscar; si es igual el valor fue encontrado, en caso contrario si el elemento a buscar es menor se realiza la búsqueda en los elementos de la izquierda y si es mayor al elemento medio se busca a la derecha del mismo.

Búsqueda lineal o secuencial. Es la manera más sencilla de buscar un elemento en un vector. Se recorre desde el primer elemento hasta el último, de uno en uno. Si se encuentra el elemento, el resultado será la posición del elemento buscado y si no aparecerá el mensaje respectivo o cero (es falso que se encontró).

Centinela. Variable que inicia con un valor, luego dentro de un bucle este valor cambia, haciendo falsa la condición del ciclo y por lo tanto indicará el fin del ciclo (el usuario puede determinar cuándo hacerlo). La repetición controlada por centinela se considera como una repetición indefinida (se desconoce el número de repeticiones). Por ejemplo un usuario puede introducir S o N para indicar si desea continuar o no. El bucle terminará cuando la respuesta del usuario sea "n" o "N". También se podrían manejar números.

Codificar. Se le conoce como código fuente y consiste en transcribir el algoritmo definido en la etapa de diseño en un código reconocido por la computadora, es decir en un lenguaje de programación, por ejemplo en lenguaje "C".

Comentarios. En cualquier lenguaje de programación sirven para que el código fuente sea más entendible, aumentan la claridad de un programa, ayudan a la documentación y bien utilizados nos pueden ahorrar mucho tiempo. Son útiles para identificar los elementos principales de un programa o explicar la lógica subyacente de éstos. Deben ser breves y evitar ambigüedades.

Constante. Dato que permanece sin cambio durante el desarrollo del algoritmo o durante la ejecución del programa, es decir valores fijos que no pueden ser alterados por el usuario.

Contador. Es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción en cada repetición; dicha variable controla o determina la cantidad de veces que se repite un proceso o dato. Es una forma de controlar a un bucle.

Conversión de tipos. Cuando se evalúa una expresión aritmética que contiene diferentes tipos de datos, el compilador convierte todos ellos a un tipo único; se lleva a cabo operación a operación. Estas conversiones pueden aumentar o disminuir la precisión del tipo al que se convierten los elementos de la expresión.

Desde (*for*). Es la estructura repetitiva más utilizada y simple de manejar, ya que repite un conjunto de instrucciones un número determinado de veces. Una de sus aplicaciones principales son los arreglos.

Diagrama de flujo. Representación gráfica de un algoritmo. Dicha representación se lleva a cabo cuando los símbolos (que indican diferentes procesos en la computadora) se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar las instrucciones para obtener los resultados deseados.

Estructura de control repetitiva o de iteración condicional. La repetición de una acción (una o varias instrucciones) se lleva a cabo mientras se cumpla cierta condición; para que la acción termine, la acción misma debe modificar la(s) variable(s) de control que interviene(n) en la condición. Dicha condición puede estar predefinida como en el ciclo desde (*for*); o no predeterminada, como en los bucles mientras (*while*) y hacer-mientras (*do_while*).

Estructura de control secuencial. Es la estructura más sencilla ya que el programador identifica los datos de entrada, los procesa y muestra o imprime los datos de salida.

Estructura de control selectiva o alternativa. Es una estructura con una sola entrada y una sola salida en la cual se realiza una acción (una o varias instrucciones) de entre varias, según una condición; o se realiza una acción según el cumplimiento o no de una determinada condición. La condición puede ser simple o compuesta.

Etapas o pasos en la creación de un programa: 1) Definición del problema, 2) Análisis del problema, 3) Diseño y técnicas para la formulación de un algoritmo (diagrama de flujo y pseudocódigo), 4) Codificación, 5) Prueba y depuración, 6) Documentación (interna y externa), 7) Mantenimiento.

Expresión. Resultado de unir operandos mediante operadores. Los operandos pueden ser variables, constantes u otras expresiones; los operadores pueden ser aritméticos, lógicos o relacionales. El resultado de una expresión es un dato numérico o un valor lógico. Para agrupar las expresiones utilizamos los paréntesis.

Hacer_mientras (*do while*). Esta estructura le permite al programador especificar que se repita una acción en tanto cierta condición sea verdadera; cuando ésta es falsa se sale del ciclo. La condición la revisa después del ciclo o bucle.

Identificador. Secuencia de caracteres alfabéticos, numéricos y el guión bajo. Con ellos podemos dar nombre a variables, constantes, tipos de dato, nombres de funciones o procedimientos, etcétera.

Lenguaje de alto nivel. Es semejante, al lenguaje natural humano (en general en inglés), facilitando la elaboración y comprensión del programa. Por ejemplo, Basic, Pascal, Cobol, Fortan, C, etcétera.

Lenguaje de bajo nivel (ensamblador). Las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones.

Lenguaje de programación. Combinación de símbolos y reglas que permiten la elaboración de programas; con los cuales la computadora puede realizar tareas o resolver problemas de manera eficiente.

Lenguaje máquina. Las instrucciones son directamente entendibles por la computadora y no necesitan de un traductor para que la CPU pueda entender y ejecutar el programa. Utiliza el código binario (0 y 1), se basa en bits (abreviatura inglesa de dígits binarios).

Mientras (*while*). Evalúa la expresión booleana proporcionada en los paréntesis (condición), y si su valor es verdadero (distinto de cero) se realizará el ciclo o bucle (una o varias instrucciones). Después, la condición es reevaluada y se procede de la misma manera hasta que la condición se vuelve falsa.

Operador. Símbolo que permite relacionar dos datos en una expresión y evaluar el resultado de la operación.

Ordenación. Su finalidad es clasificar datos (generalmente arreglos o ficheros) en un orden ascendente o descendente mediante un criterio (numérico, alfabético...). De acuerdo con el tipo de elemento que se quiera ordenar puede ser: 1) Ordenación interna y 2) Ordenación externa.

Ordenación externa: Los datos están en un dispositivo de almacenamiento externo (ficheros), y su ordenación es más lenta que la interna.

Ordenación interna: Los datos se encuentran en memoria (arreglos, listas, etc.) y pueden ser de acceso aleatorio o directo. 1) Directos: Burbuja, selección, inserción directa, etc. 2) Indirectos (avanzados): Shell, ordenación rápida, ordenación por mezcla.

Palabras reservadas. Son palabras que tienen un significado especial para el lenguaje y no se pueden utilizar como identificadores.

Parámetros actuales o reales. Pueden ser variables o expresiones cuyos valores se pasan a los argumentos formales. Se emplean en la llamada a la función. Algunos autores los llaman argumentos.

Parámetros formales o ficticios. Son variables de la función que toman los valores que se le pasan desde fuera de ella. Son los que se encuentran en la definición de la función. Si una función usa argumentos, éstos se comportan como variables locales de la función. La declaración de una función se conoce también como prototipo de la función.

Parámetros o argumentos de una función. Valores que nos permiten la comunicación entre dos funciones. Corresponden con una serie de valores que se especifican en la llamada a la función, o en la declaración de la misma, de los que depende el resultado de la función.

Parámetros por valor. En cada llamada a la función se genera una copia de los valores de los parámetros actuales, que se almacenan en variables temporales en la pila mientras dure la ejecución de la función.

Parámetros por variable o por referencia. En este tipo de parámetros los datos salen modificados. Se utilizan cuando una función modifica el valor de una variable que se pasa como parámetro actual en la llamada a la función. Se proporciona a la función la dirección de la variable, lo cual se realiza mediante un puntero que señale a esa dirección.

Programa de computadora. Es un algoritmo desarrollado en un determinado lenguaje de programación, para ser utilizado por la computadora. Es decir, son una serie de pasos o instrucciones ordenados y finitos que pueden ser procesados por una computadora, permitiéndonos resolver un problema o tarea específica.

Programación estructurada. Es un paradigma o forma de programar; consiste en un conjunto de técnicas que nos permiten desarrollar programas fáciles de escribir, verificar, leer y mantener. Incluyen: 1) Diseño descendente (top-down), 2) Estructuras de datos, 3) Estructuras de control, 4) Programación modular.

Programación modular. Técnica que consiste en dividir un programa en tareas que dan origen a la creación de pequeños programas llamados módulos, subprogramas o subrutinas con la finalidad de simplificar la elaboración y mantenimiento del mismo. Cada módulo se codifica y se procesa de manera independiente, sin preocuparse de los detalles de otros módulos.

Pseudocódigo. Se puede definir como un lenguaje de especificaciones de algoritmos. Es una mezcla del lenguaje natural (español, inglés o cualquier otro idioma), símbolos y términos utilizados en la programación.

Recursividad. Es cuando una función se llama a sí misma. Para finalizar la recursividad debe existir una condición previamente definida.

Registro o estructura. Es un tipo de dato estructurado y definido por el usuario que permite almacenar datos de diferente tipo en una sola variable; dichos datos pueden ser de tipo simple (caracteres, números enteros o de coma flotante etc.) o a su vez de tipo compuesto (vectores, estructuras, listas, etc.). A los datos del registro se les denomina campos, elementos o miembros. Una estructura está formada por variables que tienen relación entre sí.

Selectiva doble *si/si-no* (*if/else*). Dirige a la computadora para ejecutar una acción si la condición es verdadera, y otra acción en caso de que sea falsa. Las instrucciones deberán ser diferentes en cada caso, ya que si fueran iguales no se requeriría una estructura selectiva.

Selectiva múltiple *segun_sea* (*switch*). Selecciona entre varias alternativas, dependiendo del valor de la expresión. Cuando en la estructura si (*if*) todas las condiciones utilizan la igualdad de una variable o expresión determinada con constantes predefinidas, se puede utilizar la instrucción *segun_sea* (*switch*); en dicha instrucción existen más de dos opciones. La variable o expresión sólo acepta valores enteros o caracteres para entrar a la opción y el operador de relación es el igual.

Selectiva simple *si* (*if*). Dirige a la computadora para ejecutar una o más instrucciones solamente si la condición es verdadera. Si la condición es falsa no realiza ninguna acción.

sizeof. Proporciona la cantidad de memoria ocupada por los elementos de datos, *sizeof* indica el número de bytes utilizado por cualquier variable particular en un determinado compilador.

Tipos de datos. Los diferentes objetos de información con los que un algoritmo o programa trabaja se conocen colectivamente como datos. Todos los datos tienen un tipo asociado con ellos; el tipo de un dato es el conjunto (rango) de valores que puede tomar durante el programa. El tipo de un dato determina la naturaleza del conjunto de valores que puede tomar una variable.

Variable. Dato cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Es decir, representará un valor almacenado en memoria que se puede modificar en cualquier momento o conservar para ser usado tantas veces como se desee.

Variable global. Variable declarada fuera de cualquier función y puede ser utilizada por las funciones que se encuentran después de dicha declaración; por lo tanto si la declaramos junto a las librerías, la podrá utilizar todo el programa. Esta característica es propia de lenguaje C.

Variable local. Variable declarada en una determina función, que sólo se encuentra disponible durante el funcionamiento de la misma, es decir está en memoria cuando dicha función está activa.

Glosario de funciones en lenguaje C

En esta sección describiremos sólo las funciones utilizadas en el presente libro.

Palabras reservadas de ANSI C

auto	*break*	*case*	*char*	*const*	continue	*default*	*do*
double	*else*	enum	extern	*float*	*for*	goto	*if*
int	*long*	register	*return*	*short*	signed	tamof	static
struct	*switch*	*typedef*	union	unsigned	*void*	volatile	*while*

break. Se utiliza en un *switch* para salir del *case* y con ello terminar dicha función *switch*. Se usa también dentro de los ciclos (*while*, *do-while*, *for*) para indicar que cuando llega a esa línea de código, sale del bucle respectivo.

case. A la derecha aparece un valor constante. Cuando se encuentra la función case que concuerda con el valor del *switch* se ejecutan la(s) sentencia(s) que le siguen hasta encontrar un *break* y salir del *switch*.

char. Tipo de dato para manejo de caracteres de 8 bits con rango de –128 a 127.

const. Nos permite declarar constantes al igual que *#define*.

default. Conjunto de sentencias que se ejecutarán si el valor de la expresión no coincide con ninguna de las constantes evaluadas por el *switch*.

do. Ciclo condicional que se utiliza con la instrucción *while*, generando el ciclo *do-while*. El ciclo se ejecuta por lo menos una vez, ya que la condición la evalúa al final.

double. Número en coma flotante (real) de doble precisión. Tiene 64 bits y rango de 1.7E−308 a 1.7E+308. Con una precisión hasta de 16 dígitos.

else. Bloque que se ejecuta cuando en la estructura selectiva doble (*if-else*) la condición es falsa.

flota. Número en coma flotante (real) de doble precisión. Tiene 32 bits y rango de 3.4E−38 a 3.4E+38. Con una precisión de 15 dígitos.

for. Bucle que se ejecuta una cantidad definida de veces.

if. Bloque que se ejecuta cuando en la estructura selectiva simple (*if*) o doble (*if-else*) la condición es verdadera.

int. Número entero con signo de 16, 32 o 64 bits. En sistemas de 16 bits su rango de valores es de –32763 a 32762. Para sistemas de 32 bits el rango es de –2147483648 a 2147483647.

long. Número entero de 32 bits de rango igual a –2147483648 a 2147483647.

return. Especifica el dato que devuelve una función.

struct. Define una estructura.

switch. Estructura alternativa múltiple, que por medio de una expresión permite seleccionar un bloque de instrucciones a ejecutar de entre varios posibles.

typedef. Define un tipo de dato. En lenguaje C podemos dar un nuevo nombre a tipos de datos que ya existen y que son más afines con aquello que representan.

void. Indica que una función no devuelve valor alguno o no tiene parámetros.

while. Bucle que se ejecuta mientras la condición sea cierta, dicha condición se evalúa antes de entrar al ciclo.

Entrada y salida formateada (vea el anexo A)

printf(). Se usa para la salida, nos permite escribir textos y datos en la pantalla con determinado formato.

scanf(). Se usa para la entrada, nos permite leer (introducir) datos desde el teclado y asignarlos a variables de C, de acuerdo con el formato especificado.

Entrada y salida sin formato (vea el anexo A)

getch(). Lee un carácter del teclado, no hay que esperar que se pulse la tecla <enter>. No visualiza el eco del carácter.

getchar(). Lee un carácter del teclado, espera hasta que se pulsa la tecla <enter>.

getche(). Lee un carácter del teclado, no hay que esperar hasta que se pulse la tecla <enter>. Visualiza el eco del carácter.

gets(). Lee una cadena de caracteres introducida por el teclado.

putchar(). Imprime un carácter en la pantalla en la posición actual del cursor.

puts(). Imprime una cadena en la pantalla, seguida de un carácter de salto de línea.

Funciones matemáticas (librería math.h)

abs(x). Valor absoluto de un int.

acos(x). Arco coseno de x.

asin(x). Arco seno de x.

atan(x). Arco tangente de x.

ceil(x). Redondea hacia el número inmediato superior.

cos(x). Coseno de x en radianes.

exp(x). Exponencial de x, e^x.

fabs(x). Valor absoluto de un double.

floor(x). Redondeo hacia el número inmediato inferior.

fmod(x,y). Residuo de x/y como número de punto flotante.

log(x). Logaritmo natural de x.

log10(x). Logaritmo en base 10 de x.

pow(x,y). Potencia x^y.

sin(x). Seno de x en radianes.

sqrt(x). Raíz cuadrada de x.

tan(x). Tangente de x en radianes.

Funciones para manejo de caracteres y cadenas

Librería *ctype*, pruebas y conversiones de caracteres

Macros

isalpha(c). Comprueba si un carácter es *alfabético*.

isdigit(c). Comprueba si un carácter es un *dígito decimal*.

islower(c). Comprueba si un carácter es de tipo *minúscula*. Devuelve un valor distinto de cero si *c* es cualquiera de las letras minúsculas [a-z].

ispunct(c). Comprueba si un carácter es *signo de puntuación*.

isspace(c). Comprueba si un carácter es un espacio.

isupper(c). Comprueba si un carácter es de tipo *mayúscula*. Devuelve un valor distinto de cero si *c* es cualquiera de las letras mayúsculas [A-Z].

toascii(c). Convierte *c* a su correspondiente código ASCII; regresa un valor dentro del rango entre 0 y 127.

Funciones

tolower(c). Devuelve la correspondiente letra minúscula si existe y si *isupper*(*c*) es distinto de cero; en caso contrario, devuelve *c*.

toupper (c). Devuelve la correspondiente letra mayúscula si existe y si *islower*(*c*) es distinto de cero; en caso contrario, devuelve *c*.

Operaciones con cadenas de caracteres, *string.h* y *stdlib.h*

Librería string.h

strcat (cad1,cad2). Concatena (agrega) *cad2* al final de *cad1* (incluyendo el carácter nulo); el carácter inicial de *cad2* sobreescribe el carácter nulo al final de *cad1*. Devuelve el valor de *cad1*.

strchr(cad,c). Localiza la primera ocurrencia del carácter *c* en la cadena *cad*; si no se encuentra, devuelve un puntero nulo.

strcmp(cad1,cad2). Compara los elementos de dos cadenas *cad1* y *cad2*. Si son iguales, devuelve 0. Si el de *cad1* es mayor que *cad2*, devuelve un valor mayor que cero; en caso contrario, devuelve un valor menor que cero.

strcpy(cad1,cad2). Copia la cadena *cad2*, incluyendo el nulo, en el arreglo *cad1*. Devuelve el valor de *cad1*.

strlen(cad). Devuelve el número de caracteres de la cadena *cad*, sin incluir el nulo de terminación.

strlwr(cad). Convierte a minúsculas la cadena *cad*.

strncat(cad1,cad2,n). Concatena *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.

strncmp(cad1,cad2,n). Igual que *strcmp* pero con *n* caracteres, compara los *n* primeros caracteres de una cadena.

strncpy(cad1,cad2,n). Copia los primeros *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.

strstr(cad1, cad2). Busca en *cad1* la subcadena *cad2*, si la encuentra devuelve un puntero hacia la primera ocurrencia de *cad2* en *cad1*, si no regresa *null*.

strupr(cad). Convierte a mayúsculas la cadena *cad*.

Funciones para la conversión de tipos, librería *stdlib.h*

atof(cad). Convierte los caracteres de la cadena *cad* a la representación interna de tipo *double* y devuelve ese valor.

atoi(cad). Convierte los caracteres de la cadena *cad* a la representación interna de tipo *int* y devuelve ese valor.

Manejo de consola, librerías *conio.h* y *conio2.h* (Dev-C++ y Code::Blocks)

clrscr(). Borra toda la pantalla. En general, esta operación sólo se aplica cuando la pantalla está en modo de texto.

gotoxy(int x, int y). Sitúa el cursor en la fila y columna definidas por *x,y*. Esta operación sólo se utiliza en pantallas de modo texto.

highvideo(). Aumenta la intensidad de la luminosidad de los caracteres. Afecta a todos los caracteres que se imprimen posteriormente en pantalla dentro de esa ventana.

kbhit(). Revisa si se presionó una tecla. Retorna 0 si no se ha pulsado una tecla.

normvideo(). Anula a *highvideo()* y/o *normvideo( )*. Es decir, la luminosidad de los caracteres que se escriban a continuación, será normal dentro de esa ventana.

textbackground(int color). Selecciona un nuevo color del fondo de la pantalla.

textcolor(int color). Selecciona un nuevo color de los caracteres que se escribirán a continuación.

window(int iniciox,int inicioy, int finx, int finy). Crea una ventana activa en pantalla.

Bibliografía y recursos de programación

Alcalde E. (1992). *Metodología de la programación*. McGraw-Hill.

Antonakos James L. y Mansfield Kenneth C. (1997). *Programación estructurada en* C. Prentice Hall.

Cairó Battistutti O. (2005). *Metodología de la programación: algoritmos, diagramas de flujo y programas.* 3a. edición. Alfaomega.

Cairó Battistutti O. (2006). *Fundamentos de programación. Piensa en C.* Pearson.

Ceballos Sierra F. J. (2006). *Enciclopedia del lenguaje C.* 8a. reimpresión. México. Alfaomega.

Ceballos Sierra F. J. (2007). *C/C++ Curso de programación.* 4a. edición. Alfaomega Ra-Ma.

Criado Clavero Ma. A. (2006). *Programación en lenguajes estructurados.* Alfaomega Ra-Ma.

Deitel Harvey M. (2004). *Cómo programar en C++.* 4a. edición. Pearson.

Deitel Harvey M. y Deitel Paul J. (2004). *C/c++ Cómo programar y Java* 4a. edición. Prentice Hall.

García Bermejo. J. R. (2008). *Programación estructurada en C.* Pearson.

García Carballeira F. y Calderón Mateos A. (2002). *Problemas resueltos de programación en lenguaje C.* Thomson.

García Molina J. J., Montoya Dato F., Fernández Alemán J. L., y Majado Rosales Ma. J. (2005). *Una introducción a la programación. Un enfoque algorítmico.* Thomson.

Gottfried Byron (2005). *Programación en C.* 2a. ed. McGraw-Hill.

Joyanes Aguilar L. (2000). *Problemas de metodología de la programación.* McGraw-Hill.

Joyanes Aguilar L. y Zahonens Martínez I. (2001). *Programación en C: Metodología, estructura de datos y objetos.* España. McGraw-Hill.

Joyanes Aguilar L. (2008). *Fundamentos de programación*. 4a. edición. McGraw-Hill.

Joyanes Aguilar L. (2000). *Programación en C++ algoritmos, estructuras de datos y objetos*. McGraw-Hill.

Joyanes Aguilar L. (2000). *Programación en C++.* Madrid. McGraw-Hill.

Joyanes Aguilar L. (2003). *Fundamentos de programación. Algoritmos, estructuras de datos y objetos.* 3a. ed. Madrid. McGraw-Hill.

Joyanes Aguilar L. (2003). *Programación orientada a objetos*. 2a. edición. Madrid. McGraw-Hill.

Joyanes Aguilar L. y Sánchez García L. (2006). *Programación en C: un enfoque práctico*. Serie Schaum. Madrid. McGraw-Hill.

Joyanes Aguilar L. y Castillo Sanz A. (2002). *Programación en C: Libro de problemas*. McGraw-Hill.

Joyanes Aguilar L. y Sánchez García L. (2006). *Programación en C++. Algoritmos, estructuras de datos y objetos*. 2a. ed. Colección Schaum. Madrid. McGraw-Hill.

Joyanes Aguilar L., Rodríguez Baena L. y Fernández M. (2003). *Fundamentos de programación*. Libro de problemas. 2a. edición. Madrid. McGraw-Hill.

Joyanes Aguilar L. y Zahonero Martínez I. (2004). *Algoritmos y estructuras de datos. Una perspectiva en C.* McGraw-Hill.

Joyanes Aguilar L. y Zahonero Martínez I. (2005). *Programación en C. Metodología, algoritmos y estructura de datos* L. McGraw-Hill.

Kernighan Brian W. y Ritchie Dennis M. (1991). *El lenguaje de programación* C. 2a. edición. Prentice Hall.

López Román L. (2003). *Programación estructurada: Un enfoque algorítmico*. México. Alfaomega.

Moldes Teo, F. Javier (2001). *Guía práctica para usuarios. Lenguaje C.* Anaya Multimedia.

Peña Marí R. (2005). *Diseño de programas. Formalismo y abstracción.* 3a. ed. Pearson Educación.

Peñolaza Romero, E. (2004). *Fundamentos de programación C/C++*. 4a. ed. Alfaomega.

Ramírez, F. (2007). *Introducción a la programación: algoritmos y su implementación en Visual Basic.* NET, C #, Java y C++ (2a. ed.). México. Alfaomega.

Santos M., Patiño I. y Carrasco R. (2005). *Fundamentos de programación.* Editorial Ra-Ma.

Schildt, Herbert (2001). C. *Manual de referencia.* 4a. ed. Osborne/McGraw-Hill.

Tutoriales

Página de Dennis M. Ritchie: http://www.cs.bell-labs.com/who/dmr/index.html

http://c.conclase.net/borland/

http://www.monografias.com/trabajos4/lenguajec/lenguajec.shtml

http://html.rincondelvago.com/lenguaje-de-programacion-c_1.html

http://sopa.dis.ulpgc.es/so/cpp/intro_c

http://webpages.ull.es/users/fsande/talf/cursoc

http://www.carlospes.com/curso_de_lenguaje_c

http://www.monografias.com/trabajos38/programacion/programacion.sthml

http://www.monografias.com/trabajos38/programacion/programacion.sthml

http://www.nachocabanes.com/c/index.php

http://algoritmosydiagramassosa.blogspot.com/

http://www.uhu.es/04004/material/Transparencias4.pdf

http://silvercorp.wordpress.com/2006/09/28/lenguaje-c-ejemplos-basicos/

http://mimosa.pntic.mec.es/~flarrosa/lengc.pdf

http://sicuz.unizar.es/siscen/doc/c-funciones.pdf

http://www.algoritmia.net/articles.php?id=31

Compiladores

http://www.bloodshed.net

http://www.codeblocks.org



Índice analítico

P

R

S

T

U

V